intel

EMV-51A
EMULATION VEHICLE
USER'S GUIDE

Copyright 1984, Intel Corporation
Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051

Order Number: 165624-001

EMV-51A
EMULATION VEHICLE
USER'S GUIDE

Order Number: 165624-001

Copyright 1984, Intel Corporation
Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051

CAUTION

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instruction manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A Computing Device pursuant to Subpart J of Part 15 of FCC rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the user, at his own expense, will be required to take whatever measures may be required to correct the interference.

Additional copies of this manual or other Intel literature may be obtained from:

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

The information in this document is subject to change without notice.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update nor to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's software license, or as defined in ASPR 7-104.9(a)(9).

No part of this document may be copied or reproduced in any form or by any means without prior written consent of Intel Corporation.

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel Corporation retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products:

BITBUS	im	iRMX	Plug-A-Bubble
COMMputer	iMDDX	iSBC	PROMPT
CREDIT	iMMX	iSBX	Promware
Data Pipeline	Insite	iSDM	QueX
Genius	Intel	iSXM	QUEST
i	intel	KEPROM	Ripplemode
i	intelBOS	Library Manager	RMX/80
I²ICE	Intelevision	MCS	RUPI
ICE	inteligent Identifier	Megachassis	Seamless
iCS	inteligent Programming	MICROMAINFRAME	SLD
iDBP	Intellec	MULTIBUS	SYSTEM 2000
iDIS	Intellink	MULTICHANNEL	UPI
iLBX	iOSP	MULTIMODULE	
	iPDS	OpenNET	

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

*MULTIBUS is a patented Intel bus.

Copyright 1984, Intel Corporation

REV.	REVISION HISTORY	DATE
-001	Orignal Issue	12/84

PREFACE

This manual provides assistance for using the EMV-51A emulation vehicle. Chapter 1 contains an overview of the EMV-51A components and the emulation process. Chapter 2 provides the installation instructions and a tutorial to familiarize the initial user with the emulator commands. Chapter 3 contains a technical description of the EMV-51A. Chapters 4 through 8 describe the EMV-51A emulator software commands. Chapter 9 contains reference material about the EMV-51A and the MCS® -51 family of microprocessors. Chapter 10 contains an alphabetical command dictionary. Appendix A contains a listing of the error messages. Appendix B describes the confidence tests.

The following is a brief guide to the contents of the manual:

Chapter 1	includes a description of the hardware and software components of the EMV-51/51A. The chapter defines emulation and how emulation is used in the product development cycle.
Chapter 2	contains the installation instructions and provides a tutorial to help the user become familiar with the emulation procedure and some of the basic emulation commands.
Chapter 3	includes a technical description the EMV-51A components and the information needed for user-specified applications.
Chapter 4	contains an overview of the command categories used in the emulator software.
Chapter 5	presents a tutorial session on using the utility commands. It includes a description of each command and gives screen displays showing how each command is used.
Chapter 6	presents a tutorial session on using the display/modify commands. It includes a description of each command and gives screen displays showing how each command is used.
Chapter 7	presents a tutorial session on using the emulation commands. It includes a description of each command and gives screen displays showing how each command is used.
Chapter 8	presents a tutorial session on using the advanced commands. It includes a description of each command and gives screen displays showing how each command is used.
Chapter 9	contains reference material on the MCS-51 family of microprocessors and the EMV-51A.
Chapter 10	contains an alphabetical listing of the EMV-51A commands with the format requirements.
Appendix A	lists the error and warning messages.
Appendix B	contains instructions for running the EMV-51A confidence tests.

The reader should reference the following material to obtain additional information about the MCS-51 family of microprocessors:

Microcontroller Handbook, order number 210918
MCS®-51 Macro Assembler User's Guide, order number 9800937

For further information regarding Intel literature, courses, and the maintenance agreement, the following literature has been assembled in a single packet, order number 121584-001.

- *Literature Guide*
- *Intel Workshops Catalog*
- *Peace of Mind: Intel's Maintenance Agreement*

SERVICE AND REPAIR ASSISTANCE

The best service for your Intel product is provided by an Intel Customer Engineer. These trained professionals provide prompt, efficient, on-site installation, preventive maintenance, and corrective maintenance services required to keep your equipment in the best possible operating condition.

The Intel Customer Engineer provides the service needed through a prepaid service contract or on an hourly charge basis. For further information, contact your local Intel sales office.

When the Intel Customer Engineer is not available, contact the Intel Product Service Center.

United States customers can obtain service and repair assistance from Intel Corporation by contacting the Intel Product Service Center in their local area. Customers outside the United States should contact their sales source (Intel Sales Office or Authorized Distributer) for service information and repair assistance.

Before calling the Product Service Center, have the following information available:

1. The date you received the product.
2. The complete part number of the product (including dash number). On boards, this number is usually silk-screened onto the board. On other MCSD products, it is usually stamped on a label.
3. The serial number of the product. On boards, this number is usually stamped on the board. On other MCSD products, the serial number is usually stamped on a label mounted on the outside of the chassis.
4. The shipping and billing address.
5. If the Intel Product warranty has expired, a purchase order number is needed for billing purposes.
6. Be sure to advise the Center personnel of any extended warranty agreements that apply.

Use the following telephone numbers for contacting the Intel Product Service Center:

Western Region call: (602) 869-4951

Midwest Region call: (602) 869-4392

Eastern Region call: (602) 869-4045

International call: (602) 869-4862

Always contact the Product Service Center before returning a product to Intel for repair. You are given a repair authorization number, shipping instructions, and other important information which helps Intel provide you with fast, efficient service. If you are returning the product because of damage sustained during shipment, or if the product is out of warranty, a purchase order is required before Intel can initiate the repair.

If available, use the original factory packaging material when preparing a product for shipment to the Intel Product Service Center. If the original packaging material is not available, wrap the product in a cushioning material such as Air Cap SD-240, manufactured by the Sealed Air Corporation, Hawthorne, N.J. Securely enclose it in a heavy-duty corrugated shipping carton, mark it "FRAGILE" to ensure careful handling, and ship it to the address specified by the Intel Product Service Center.

CONTENTS

CHAPTER 1
INTRODUCTION PAGE

Introduction 1-1
The MCS® -51 Architecture Overview 1-1
  Designing With the 8051 1-1
The EMV-51 and the EMV-51A 1-1
EMV-51A Components 1-2
  Controller 1-2
  Emulator 1-4
  Software 1-4
  Publications 1-5
Emulation Overview 1-5
  Definition of Emulation 1-5
  Without Emulation 1-6
  With Emulation 1-6
    Symbolic Debugging 1-7

CHAPTER 2
GETTING STARTED

Installation Procedures 2-1
  Hardware Installation 2-1
  Software Installation 2-4
Emulation Session 2-6

CHAPTER 3
TECHNICAL DESCRIPTION

Introduction 3-1
Controller 3-1
  Emulator 3-2
    Port 0 in Stand-Alone Mode 3-2
    Jumper Configurations 3-2

CHAPTER 4
USING THE EMV-51A

Introduction 4-1
Command Categories 4-1
Entering Commands 4-1
  Command Entry from the Console 4-2
    Continuation Lines 4-2
    Comment Lines 4-2
Entry Editing and Display Control 4-2
  Command Line Editing 4-2
Controlling the Display 4-3
  Command Entry From Files 4-3
Prompts and Messages 4-3
Abbreviated Commands 4-4

CHAPTER 5
UTILITY COMMANDS

Introduction 5-1
EMV-51A Invocation 5-1
EXIT Command 5-2
HELP Command 5-2

PAGE

  Help Information Screen Displays 5-2
LOAD Command 5-4
  Loading Multiple Programs 5-4
SAVE Command 5-4
LIST Command 5-5
Symbolic Reference Commands 5-6
System-Defined Symbols 5-6
User-Defined Symbols 5-6
DEFINE Command 5-7
Changing Symbol Values 5-7
Displaying Symbol Names 5-7
  Displaying Multiply Defined Names 5-8
SYMBOLS Command 5-8
REMOVE Command 5-8
  Enabling/Disabling Symbol Display 5-8
ENABLE Symbolic Expansion Command 5-8
DISABLE Symbolic Expansion Command 5-9
RESET Command 5-10
The SUFFIX and BASE Commands 5-10
EVALUATE Command 5-11

CHAPTER 6
DISPLAY/MODIFY COMMANDS

Introduction 6-1
User Register Commands 6-1
  Displaying Register Contents 6-1
  Modifying Register Contents 6-1
  The REGISTER Command 6-2
    Special Function Key 4 6-3
    Displaying INTERRUPT Register Contents 6-3
  INTERRUPT Command 6-3
Memory Commands 6-4
  Displaying Memory Contents 6-4
  Modifying Memory Contents 6-5
  Displaying Memory Commands 6-5
Assembly/Disassembly Commands 6-6
  The ASM Command 6-6
  The DASM Command 6-7
  Using DISPLAY/MODIFY Commands 6-8

CHAPTER 7
USING EMULATION COMMANDS

Introduction 7-1
Setting Breakpoints 7-1
  Address Breakpoints 7-1
  Branch Breakpoints 7-2
  Range Breakpoints 7-2
  Register Value Breakpoints 7-2
  Clearing Breakpoints 7-2

PAGE

Breakpoint Restrictions ... 7-3
Using the BREAK Command ... 7-3
Special Function Key 2 ... 7-4
Using the Trace Display ... 7-4
The TD Command ... 7-5
The TR Command ... 7-5
The TB*n* Command ... 7-5
The TS Command ... 7-5
Controlling the Trace Display ... 7-6
The TR*n* Command ... 7-6
The TV Command ... 7-6
The DTRACE Command ... 7-7
Special Function Key 3 ... 7-8
Program Emulation Commands ... 7-8
The STEP Command ... 7-8
The GO Command ... 7-9
The P Command ... 7-10
Emulation Control Screen Displays ... 7-10

CHAPTER 8
USING ADVANCED COMMANDS

Introduction ... 8-1
Using Macros ... 8-1
Defining Macros ... 8-1
The DEFINE Command ... 8-1
The EM Command ... 8-2
Parameter Passing ... 8-2
Writing Messages ... 8-2
Displaying Macro Information ... 8-3
The DIR Command ... 8-3
The MACRO Command ... 8-4
Saving Macros (the PUT Command) ... 8-4
Executing Macros ... 8-5
The Macro Execution Command ... 8-5
The ENABLE/DISABLE Macro Expansion Commands ... 8-6
Deleting Macros (the REMOVE Command) ... 8-6
Command Files ... 8-7
Executing Command Files ... 8-7
Executing Macro Command Files ... 8-7
Using Compound Commands ... 8-8
Setting Up the Conditional IF Commands ... 8-8
Setting Up the REPEAT Command ... 8-9
Setting up the COUNT Command ... 8-10
Using Special Function Keys ... 8-11
Assigning Function Keys to Macros ... 8-11
Function Key Precautions ... 8-11
Displaying Assigned Function Keys ... 8-12
Advanced Commands Demonstration ... 8-12

CHAPTER 9
EMV-51A REFERENCE MATERIAL

Introduction ... 9-1
EMV-51A Symbols ... 9-1

PAGE

EMV-51A Operators ... 9-2
Arithmetic Operators ... 9-2
Content Operators ... 9-2
Code Memory (CBYTE) ... 9-2
CBYTE Restrictions ... 9-3
On-Chip Data and Register Memory (DBYTE, RBIT, and RBYTE) ... 9-3
External Data Memory (PBYTE) ... 9-3
Relational Operators ... 9-6
Logical Operators ... 9-6

CHAPTER 10
COMMAND DICTIONARY

Introduction ... 10-1
Entering Commands ... 10-1
Specifying the Workfile ... 10-1
Command Line Execution ... 10-2
Continuation Lines ... 10-2
Entry Editing and Display Control ... 10-2
Command Line Editing ... 10-2
Controlling the Display ... 10-3
Form of EMV-51A Commands ... 10-3
Notational Conventions ... 10-3
Command Description Formats ... 10-4
Special Command Format Terms ... 10-4
Expression ... 10-5
String ... 10-6
Source ... 10-6
Destination ... 10-6
Command ... 10-6
Condition ... 10-6
Alphabetical Listing of Commands ... 10-6
ACCUMULATOR ... 10-7
ASM ... 10-8
B ... 10-9
BASE ... 10-10
BC ... 10-11
BRB ... 10-12
BREAK ... 10-13
BRR ... 10-14
BR*n* ... 10-15
BV ... 10-16
CBYTE ... 10-17
CDUMP ... 10-18
COUNT ... 10-19
DASM ... 10-21
DBYTE ... 10-22
DDUMP ... 10-23
DEFINE ... 10-24
DIR ... 10-25
DISABLE ... 10-26
DPTR ... 10-27
DTRACE ... 10-28
ENABLE ... 10-29
EVALUATE ... 10-30
EXIT ... 10-31
FUNCTION ... 10-32

PAGE

GO 10-34
HELP 10-35
IF..THEN..ELSE 10-36
INCLUDE 10-39
INTERRUPT 10-40
LIST 10-41
LOAD 10-42
MACRO 10-43
MEMORY 10-44
P 10-45
PBYTE 10-46
PC 10-47
PSW 10-48
PUT 10-49
RBIT 10-50
RBS 10-51
RBYTE 10-52
REGISTER 10-53
R*n* 10-54
REMOVE 10-55
REPEAT 10-56
RESET 10-58
SAVE 10-59
SP 10-60
STEP 10-61
SUFFIX 10-63
SYMBOLS 10-64
TB*n* 10-65
TD 10-66
TM*n* 10-67

PAGE

TR 10-68
TR*n* 10-69
TS 10-70
TV 10-71
WRITE 10-72
(macro execution) 10-73
(symbol handling) 10-74

APPENDIX A
ERROR MESSSAGES

APPENDIX B
CONFIDENCE TESTS

Confidence Test B-1
PCONF Command B-1
INIT E51CON Command B-1
Confidence Test Commands B-2
CLEAR B-2
DESCRIBE B-3
ERROR B-4
EXIT B-4
IGNORE B-5
LIST B-5
RECOGNIZE B-6
SUMMARY B-7
TEST B-8
Confidence Tests B-9
Confidence Test Error Messages B-11

INDEX

TABLES

TABLE TITLE PAGE

3-1 User Test Signals 3-1
3-2 EMV-51A Emulator Jumper Descriptions 3-3
6-1 Register Keyword Names 6-1
6-2 Memory Areas and Keyword Names 6-4
9-1 System Symbols 9-1
9-2 Punctuation and Delimiter Keywords 9-1

TABLE TITLE PAGE

9-3 Arithmetic Operators 9-2
9-4 Content Operator Keywords 9-2
9-5 Bit-Addressable Register Memory Contents 9-5
9-6 Relational Operators 9-6
9-7 Logical Operators 9-6
10-1 Available Number Bases and Digits 10-5

FIGURES

FIGURE	TITLE	PAGE
1-1	EMV-51A Hardware Components	1-2
1-2	Complete Development System	1-3
1-3	EMV-51A Controller's Test Signal Locations	1-3
1-4	EMV-51A Emulator	1-4
1-5	Typical Microcomputer Development Cycle	1-6
1-6	Development Cycle With the EMV-51A Module	1-7
2-1	EMV-51A Controller Box Fastening Screws	2-2
2-2	EMV-51A Controller Box Stand-Off Screw Locations	2-2
2-3	Jumper Locations for 4K/8K RAM Selection	2-3
2-4	Installation in the iPDS™ System	2-3
2-5	Emulator Module Connected to Prototype	2-4
2-6	Listing of the Demonstration Program	2-7
3-1	EMV-51A Controller's Terminal Post Location	3-1
3-2	Port 0 Pullup Configuration	3-2
3-3	EMV-51A Emulator Jumper Locations	3-3
9-1	Internal Data Memory Organization	9-4
10-1	Format of Command Descriptions	10-5
10-2	The BREAK Command Display Table	10-13

CHAPTER 1
INTRODUCTION

Introduction

The EMV-51A emulation vehicle is an emulation tool for the microcontroller system family of 8051-based (MCS® -51) microcontrollers and is hosted on the Intel Personal Development System (iPDS™). This chapter covers the following topics:

- MCS-51 architecture overview
- Designing with the 8051
- The EMV-51 and the EMV-51A
- EMV-51A components
- Emulation overview

The MCS® -51 Architecture Overview

The MCS-51 family consists of the 8031, the 8032, the 8051, the 8052, and the 8751 microcontrollers. The 8031 and 8032 contain no internal ROM. The 8051 contains 4K of internal ROM that is programmed during the chip manufacture, 128 bytes of internal data memory, and two counter/timers. The 8052 contains 8K of internal ROM, 256 bytes of internal data memory, and three counter/timers. The 8751 contains 4K of internal eraseable programmable read-only-memory (EPROM) which the user programs with a programmable read-only-memory (PROM) programmer.

In addition, the MCS-51 family of chips contains 32 general purpose registers, 32 I/O lines, a multiple source, nested interrupt structure, a serial I/O channel, and an on-chip oscillator and clock circuit. The microcontrollers can address 64K bytes of program memory and 64K bytes of external data memory.

The *Microcontroller Handbook*, order number 210918, contains additional information about the MCS-51 family of chips.

Designing with the 8051

Due to the complexity of the MCS-51 microcontrollers, traditional design procedures using the oscilloscope and logic analyzer are not sufficient to test and debug the user prototype hardware and software system. The 8051 is a computer on a chip and requires advanced design techniques. The capability of emulating new product designs with another computer is needed to develop and debug microcontroller/microcomputer products.

The EMV-51 and the EMV-51A

The EMV-51 emulation vehicle utilizes the 8051 microcontroller and provides 4K bytes of on-chip random-access-memory (RAM).

The EMV-51A emulation vehicle also utilizes the 8051 microcontroller; however, it can provide 8K bytes of on-board random access memory (RAM). This allows you to partially emulate the 8052 by simulating 8K bytes of read-only-memory (ROM). This does not allow you to simulate the 8052's additional 128 bytes of internal data memory or its additional timer/

counter. The amount of RAM is selected by configuring a jumper pin on the circuit board in the controller box. This procedure is described in the Hardware Installation section in Chapter 2.

NOTE

The EMV-51A software release version 1.10 or later will support both the EMV-51 and the EMV-51A. However, earlier software releases will not support the EMV-51A.

EMV-51A Components

The EMV-51A emulation vehicle comprises a controller, an emulator, a cable connecting the controller and the emulator, and emulator software on a disk. The EMV-51A package also includes several publications to assist the user in operating the EMV-51A emulator. Figure 1-1 shows the hardware components.

Controller

The controller plugs into the side of the iPDS system. It connects to the emulator via an 80 inch 50-pin flat cable, as shown in Figure 1-2.

The controller provides six terminal posts that supply test signals to the user's prototype system. These signals, shown in Figure 1-3, indicate the internal state of various operations in the 8051 microcontroller. (See Chapter 3 for a complete description of the test signals.)

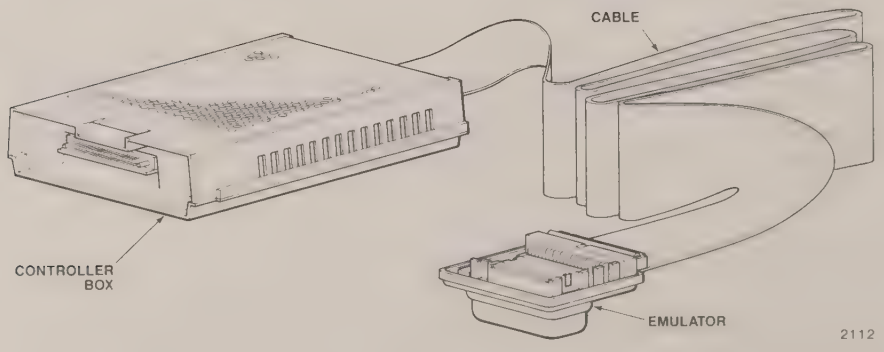


Figure 1-1 EMV-51A Hardware Components

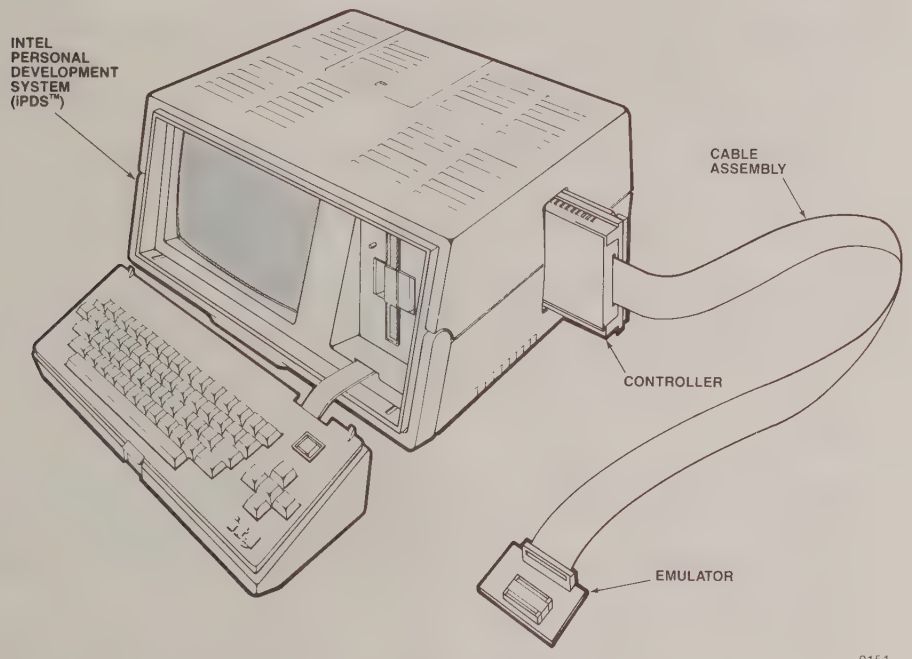


Figure 1-2 Complete Development System

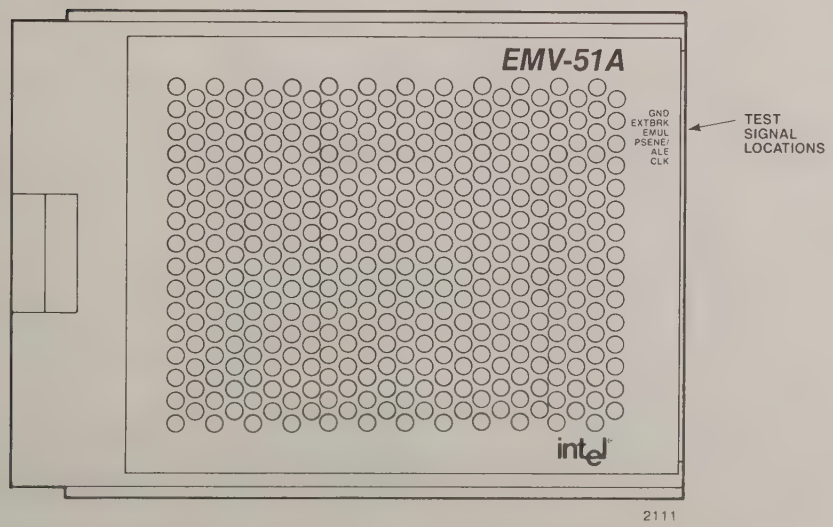


Figure 1-3 EMV-51A Controller's Test Signal Locations

Emulator

The emulator replaces the microcontroller in the user's prototype hardware system. The emulator consists of the following:

- 8051 microcontroller and integrated circuit buffers
- 12 MHz crystal
- Jumper connections for power, clock source, reset input, and the external access (EA/) pin
- 40 pins exactly like the 8051/8031/8751 microcontroller

Power for the emulator is supplied by the iPDS system or by the user's prototype system. The clock signal is supplied by a 12 MHz crystal located on the emulator circuit board or by the user's prototype system. A power source and clock signals are selected via jumper connections. Figure 1-4 is a diagram of the EMV-51A emulator.

Software

Three 5-1/4″ disks are provided with the EMV-51A package as follows:

- ISIS-PDS EMV-51A, which contains the EMV-51A emulator software
- MCS-51 software development package for the iPDS system, which contains the macro-51 assembler
- E51CE DSDD disk, which contains the diagnostic software for the EMV-51A

The ISIS-PDS EMV-51A emulator software disk contains the command file, EMV51A, and a demonstration program, EMVDEM.OBJ. The loading procedure is described Chapter 2.

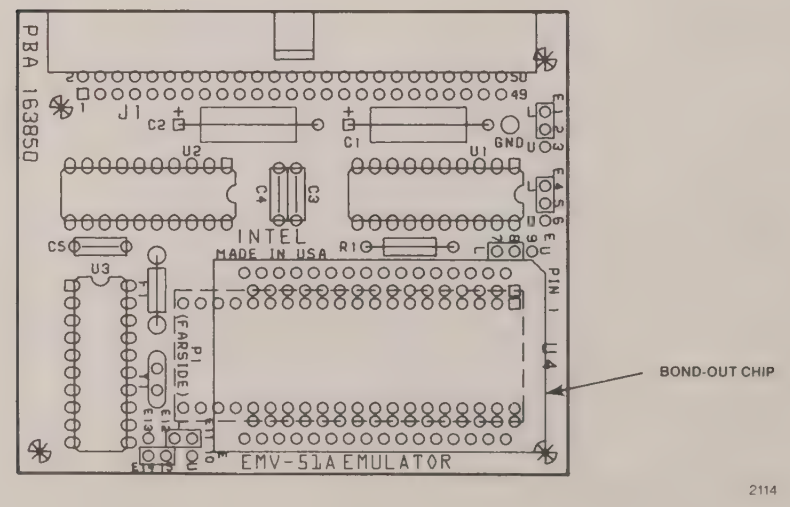


Figure 1-4 EMV-51A Emulator

Publications

The EMV-51A package includes the following instructional and reference manuals to assist the user in operating the system:

- *The EMV-51A Emulation Vehicle User's Guide*, order number 165624
- *The EMV-51A Emulation Vehicle Pocket Reference*, order number 165625
- *MCS®-51 Utilities User's Guide for 8080/8085-Based Development Systems*, order number 121737
- *MCS®-51 8048-to-8051 Assembly Language Converter Operating Instructions for ISIS-II Users*, order number 9800934
- *MCS®-51 8048-to-8051 Assembly Language Converter Operating Instructions for ISIS-II Users Change Notice*, order number 121601
- *MCS®-51 Assembler and Utilities Pocket Reference*, order number 121817
- *MCS®-51 Macro Assembler User's Guide*, order number 9800937
- *MCS®-51 Macro Assembly Language Pocket Reference*, order number 9800935

The following manuals provide additonal information and can be ordered through Intel's Literature Department:

- *Microcontroller Handbook*, order number 210918.
- *iPDS™ Personal Development System User's Guide*, order number 162606

Emulation Overview

Traditional electronics development tools, such as oscilloscopes, multimeters, and logic analyzers, are not sufficient for the development of microcontroller and microcomputer based products. These tools do not allow access to circuits and signals that are integrated onto a single silicon chip.

The EMV-51A emulation vehicle provides the tools necessary to design, develop, and debug microcontroller/microcomputer products when used with the iPDS development system.

Definition of Emulation

Emulation is the controlled execution of the user's software in a hardware environment that duplicates the microcontroller of the user's prototype system. The EMV-51A emulation vehicle creates the hardware environment for all applications using the MCS-51 family of microcontrollers.

Emulation allows interactive debugging of a user's microcontroller-based program code. Emulators permit reading and writing of microcontroller registers and system memory, program disassembly, and interactive control of program execution. The user can externally control program execution while operating in the prototype system.

The EMV-51A emulation vehicle incorporates an 8051 microcontroller that replaces the microcontroller in the user's prototype system, and acts like a window into the prototype system. With one tool, the user can watch the hardware and software of the emulator interact within the prototype system. With this capability, problems can be pinpointed very quickly.

The EMV-51A emulation vehicle connects the user's prototype system to the iPDS system. Thus, the user has complete control over the prototype system hardware. The EMV-51A emulator can contain 4K or 8K of RAM that can replace the user's ROM during emulation and permits debugging of the target system hardware and software as a unit. This procedure is described in more detail in the section entitled With Emulation later in this chapter.

Without Emulation

Prior to using emulators, the design of a new product involved a time-consuming iterative process as shown in Figure 1-5.

As Figure 1-5 indicates, each design phase progresses separately from the other. The hardware was developed and verified, then the software to control the hardware was developed and verified. The software could not be combined with the target system until both hardware and software were completely verified. If problems arose during the integration of the hardware and software, the costly design and verification phase had to be repeated until all problems were resolved.

With Emulation

When using an emulator, the product design procedure is considerably shortened. The design procedure using emulation is illustrated in Figure 1-6.

As shown in Figure 1-6, the design procedure using emulation integrates two design steps. After designing the product, construction of the target system hardware begins immediately. Another advantage of emulation is that testing and debugging the software for a product can occur before completion of the user's prototype system hardware.

When using emulation techniques, the clock and control circuit construction and verification phase can be integrated. The program code is developed and translated using the development system at the same time as construction of the target system hardware.

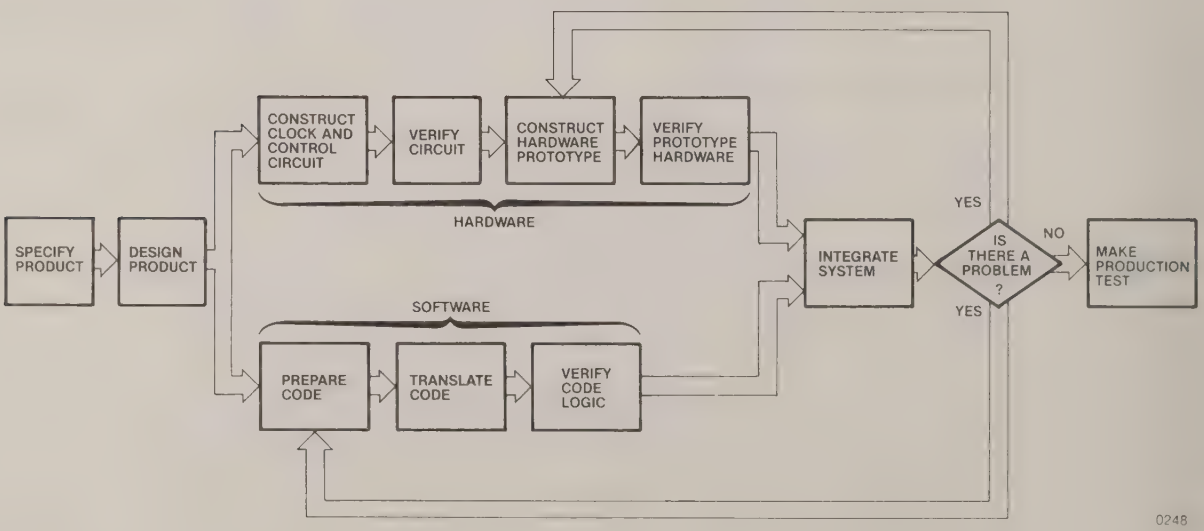


Figure 1-5 Typical Microcomputer Development Cycle

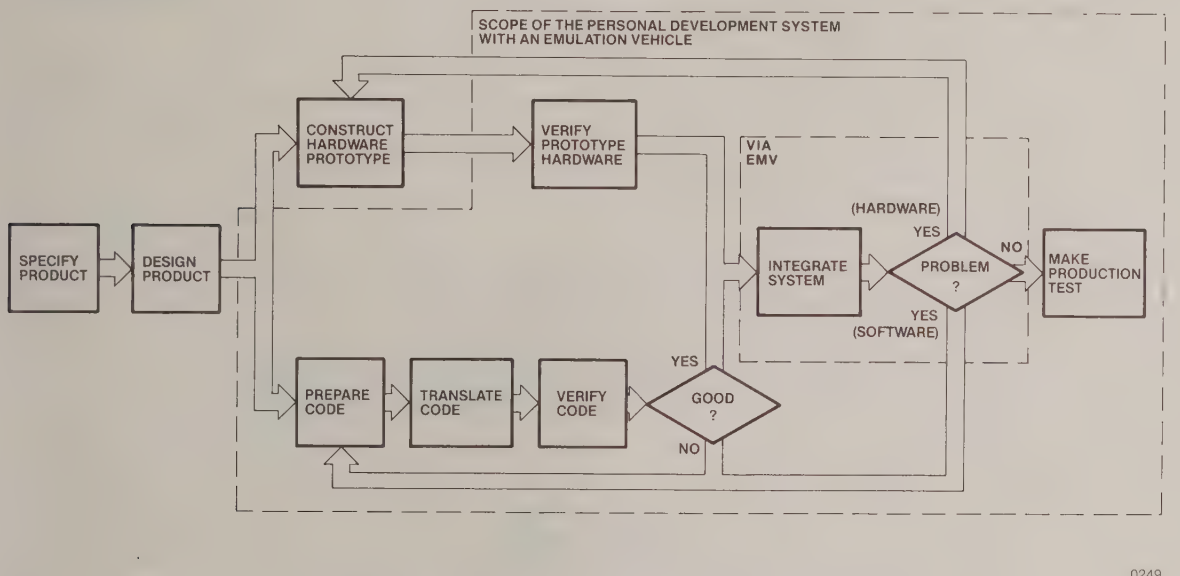


Figure 1-6 Development Cycle With the EMV-51A Module

Verification of the hardware and software can be done as a unit following construction of the target system hardware and the development of program code. This verification phase eliminates separate hardware verification, software verification, and system integration steps.

The emulator provides immediate recognition of problems between the hardware and software. When a software problem is identified, the problem is corrected with the development system and product emulation continues.

Symbolic Debugging

During emulation, reference to physical addresses and data values with labels is possible. This procedure, called symbolic debugging or symbolic referencing, improves the ability to debug the software. If the contents of the program change, the labels indicate the same addresses or data values.

Chapter 5 contains additional information about symbolic debugging. The next section presents a generalized emulation session that demonstrates some of the capabilities of the emulator.

In summary, emulation provides the following advantages:

- The controlled execution of software so the program can be monitored while being run.
- The ability to start software development prior to completion of target system hardware.
- The use of symbolic references which eliminates the need to remember physical addresses.

CHAPTER 2
GETTING STARTED

Installation Procedures

The following sections include hardware and software installation procedures.

Hardware Installation

Prior to installing the EMV-51A into the iPDS system, the plug-in module adapter board must be installed in the iPDS system. Refer to the section entitled Plug-In Module Adapter in Appendix A of the *iPDS™ User's Guide* for the installation instructions.

The following procedure is for the initial installation of the EMV-51A emulator. The selection of optional jumper configurations is covered in the section entitled Jumper Configurations in Chapter 3.

CAUTION

The 8051 bond-out chip is an MOS device which can be easily damaged by static electricity, even when the system is powered down. It is strongly advised that you perform the following steps before handling the emulator:

- Place antistatic mats on the floor
- Use a conductive surface for your workspace
- Ensure that the humidity is greater than 60%
- Use a grounding strap to ground your body to earth

1. As shipped, the EMV-51A accesses 4K bytes of RAM. To select 8K bytes of RAM, perform the following steps:

   a. Using a small Phillips screwdriver, unscrew the four screws that fasten the top of the controller to the base of the controller (see Figure 2-1).

   b. Using a 3/16″ hex nut driver, loosen the four stand-off screws with a hex pattern so that you can remove the circuit board from the controller box (see Figure 2-2).

   c. Move the jumper from E1-E3 (4K) and move it to E1-E2 (8K) (see Figure 2-3).

   d. Insert the circuit board so that components are facing the controller module cover and fasten it to the controller box using the four stand-off screws.

   e. Fasten the base of the controller box to the top of the controller box with the four Phillips screws.

2. Ensure that the power to the iPDS system is turned off.

3. Insert the EMV-51A controller from the right side of the iPDS system until it seats firmly against the connector as shown in Figure 2-4. Note that the controller module has plastic ribs along the top and bottom of the module, permitting insertion in one direction only.

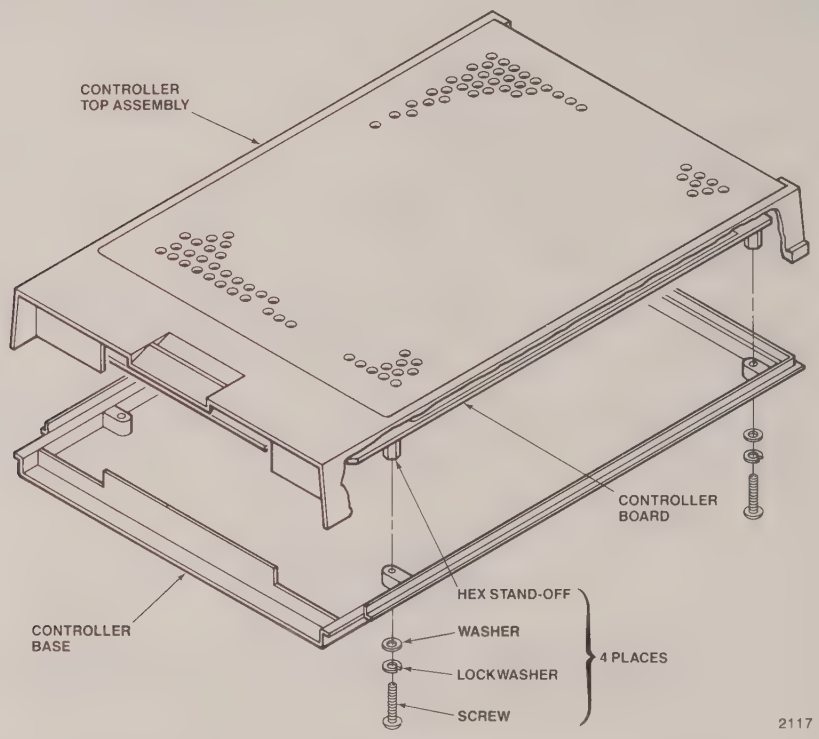


Figure 2-1 EMV-51A Controller Box Fastening Screws

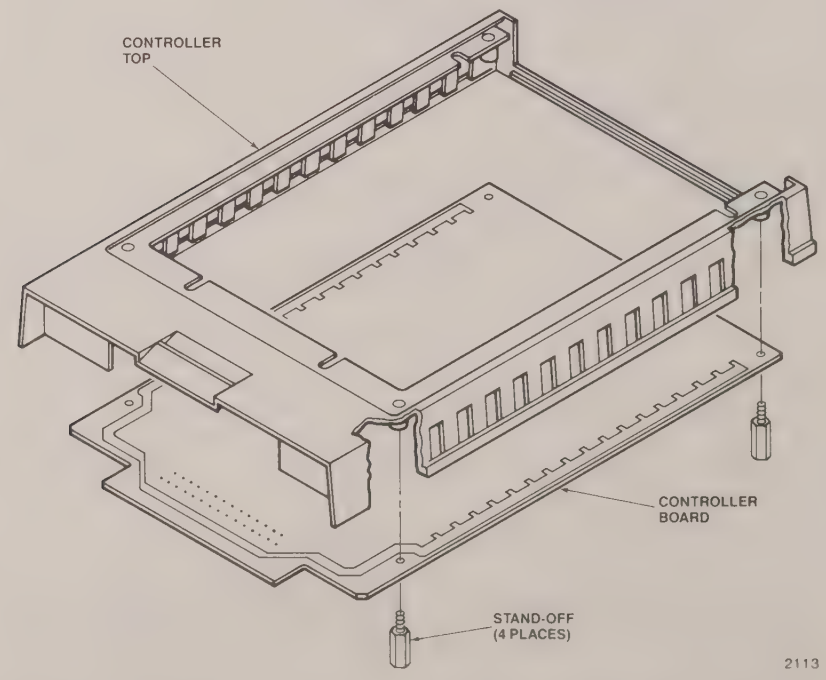


Figure 2-2 EMV-51A Controller Box Stand-Off Screw Locations

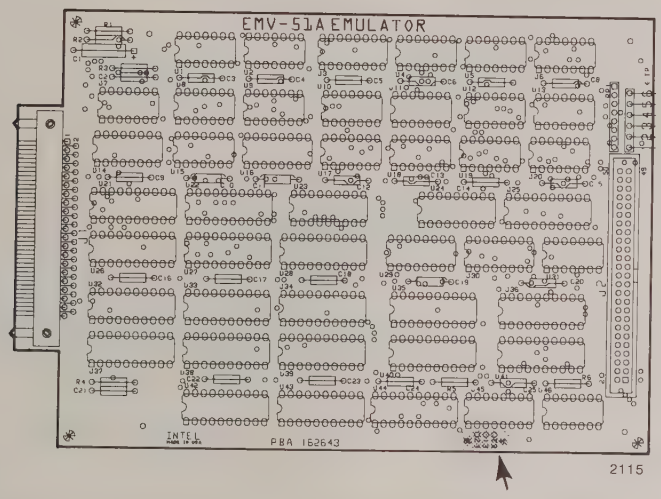


Figure 2-3 Jumper Locations for 4K/8K RAM Selection

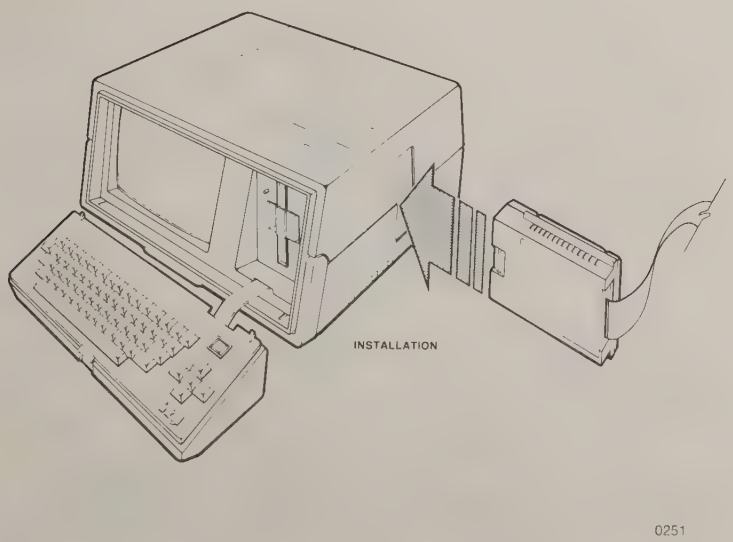


Figure 2-4 Installation in the iPDS™ System

4. Ensure proper handling of the emulator.

   a. If there is no target system available, do not remove the socket protector on the EMV-51A emulator. The EMV-51A emulator can operate without being connected to a target system.

   b. If there is a target system to be tested, remove the socket protector from the EMV-51A emulator and insert the emulator into the socket normally occupied by the 8051 or 8052 microcontroller in the target system as shown in Figure 2-5. If necessary, insert 40-pin sockets between the emulator and the target system to clear other components.

CAUTION

Do not allow objects to come in contact with the 8051 bond-out chip. Handle the user plug by the edges. If downward pressure is required when installing in the user socket, push down only on the metal strap that secures the bond-out chip in place.

Software Installation

The following procedure provides the steps necessary to install the EMV-51A software in the iPDS system.

1. Apply power to the iPDS unit.

2. Insert the iPDS system disk in drive 0.

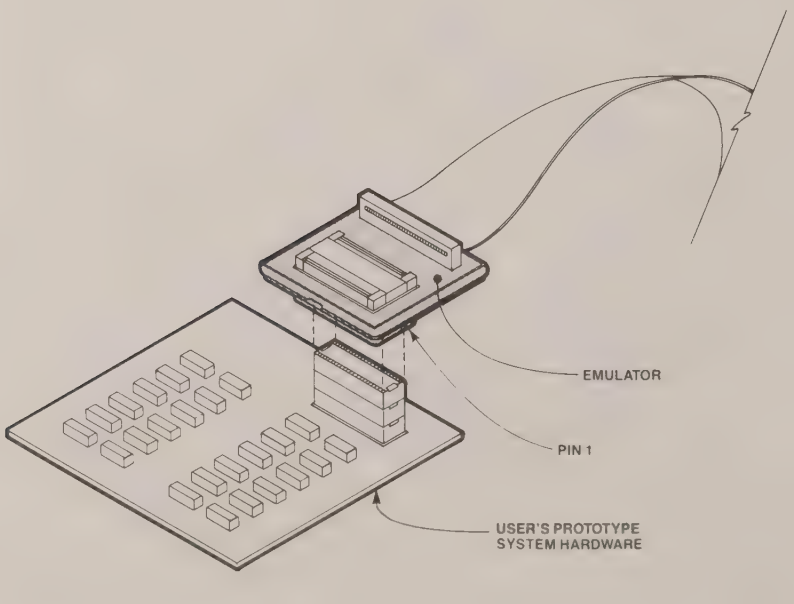


Figure 2-5 Emulator Module Connected to Prototype

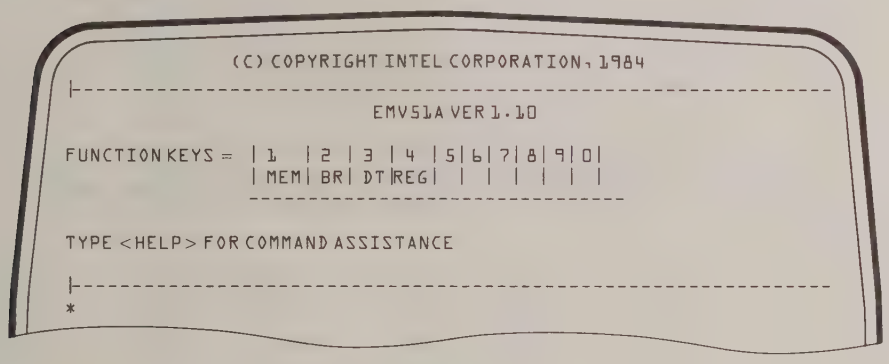
3. Either copy the ISIS-PDS EMV-51A software to the iPDS system disk or insert the ISIS-PDS EMV-51A software in an available disk drive. (Refer to the *iPDS™ User's Guide* for information on using the COPY command.)

4. Press RESET to initialize the iPDS system software and generate the sign-on message.

5. Enter the EMV51A command and obtain the sign-on message from the EMV-51A emulator software by using the following command syntax:

Key-in Sequence	Comments
:F*n*:EMV51A RETURN	The ISIS-PDS operating system loads and executes the EMV-51A software, which displays the sign-on message and the asterisk prompt shown in the screen display.

Where *n* is the number of the disk drive where the disk containing the EMV-51A software resides. (:F0: is the default disk drive and can be omitted from the command.)



NOTE

When the emulator software is loaded and executed, it selects the disk drive that the software is loaded from as the default drive for the macro workfile. The workfile is a temporary storage area on disk that holds macro definitions. The emulator software always assigns the name MAC.TMA to the workfile. The following examples illustrate how the workfile is assigned to a specific disk drive.

Key-in Sequence	Comments
EMV51A RETURN	Drive :F0: contains the EMV51A software and the workfile.
:F1:EMV51A RETURN	Drive :F1: contains the EMV51A software and the workfile.
:F2:EMV51A WORKFILE (:F1:) RETURN	Drive :F2: contains the EMV51A software and drive :F1: contains the workfile.

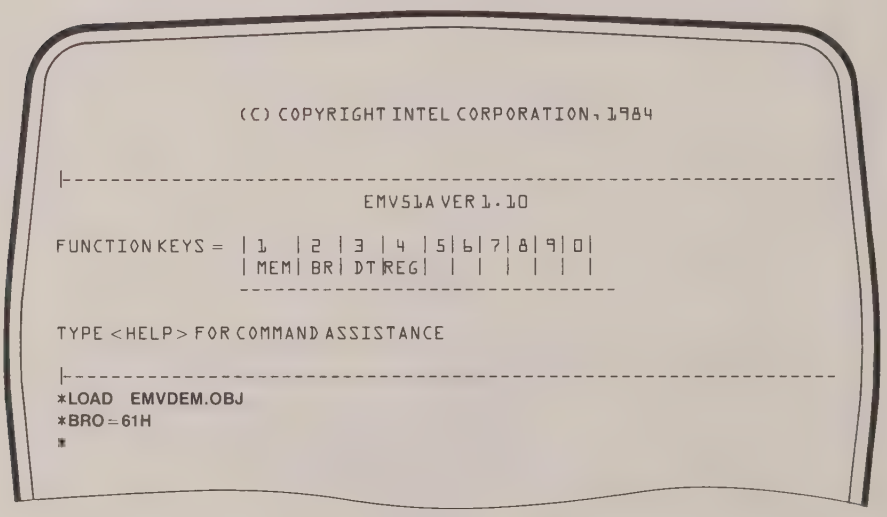
Emulation Session

1. Load the demonstration program from disk into the EMV-51A memory, using the LOAD command as follows:

Key-in Sequence	Comments
LOAD EMVDEM.OBJ RETURN	The LOAD command gets a program from a disk file and loads it into EMV-51A program memory. In this example, the program labeled EMVDEM.OBJ is loaded into memory. Figure 2-6 provides a listing of the demonstration program. The emulator software responds with the asterisk prompt when the LOAD command is completed.

2. Prepare the system for emulation by setting the emulation breakpoints as shown in the following example. A breakpoint is a specific location in a program where execution of the program should be halted. A breakpoint can be an address, a range of addresses, a register value, or a program execution branch.

Key-in Sequence	Comments
BR0 = 61H RETURN	Sets a breakpoint at address 61H.

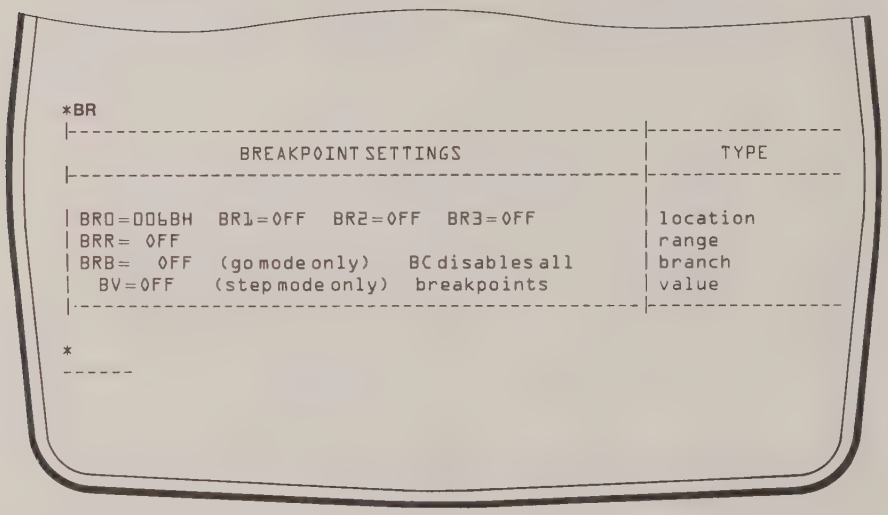

(C) COPYRIGHT INTEL CORPORATION, 1984
EMV51A VER 1.10
FUNCTION KEYS = | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 |
| MEM | BR | DT | REG | | | | | | |
TYPE <HELP> FOR COMMAND ASSISTANCE
*LOAD EMVDEM.OBJ
*BR0 = 61H
*

```
;                    EMV-51 DEMONSTRATION PROGRAM
;THIS PROGRAM ADDS TWO NUMBERS IN PACKED BCD FORMAT, AND
;CONVERTS THE SUM TO AN ASCII STRING. THE TWO NUMBERS ARE
;STORED IN CONSECUTIVE MEMORY BYTES AS:
;
;RO->      FIRST BYTE =     LEAST AND NEXT LEAST SIGNIFICANT DIGITS OF AU-
                            GEND
;          SECOND BYTE =    NEXT TWO MOST SIGNIFICANT DIGITS OF AUGEND
;          THIRD BYTE =     MOST AND NEXT MOST SIGNIFICANT DIGITS OF AU-
                            GEND
;          FOURTH BYTE =
;          FIFTH BYTE =     >SAME FOR ADDEND
;          SIXTH BYTE =     /
;
;THE RESULT IS RETURNED IN A STRING POINTED TO BY R1.
;
*************************************************************************

          ORG    0040H
          R0X    EQU  00H      ;PSEUDONYM FOR R0
DEMO;     MOV    R1,#BUFFER    ;GET POINTER TO RESULTS AREA
          MOV    R2,#03        ;ST BCD DIGITS COUNTER
          CLR    CY            ;GET READY FOR ADDITION ACTION
          PUSH   R0X           ;SAV PARAMETER POINTER
LOOP:     MOV    A,@R0         ;GET TWO BCD DIGITS FROM AUGEND
          INC    R0            ;POINT TO CORRESPONDING DIGITS IN
                               ;ADDEND
          INC    R0            ; ("INC" DOESN't AFFECT CARRY FLAG)
          INC    R0
          ADDC   A,@R0         ;ADD THE TWO TOGETHER
          DA     A             ;ADJUST FOR DECIMAL ARITHMETIC
          PUSH   ACC           ;SAVE RESULT
          ANL    A,#0FH        ;GET LOW DIGIT
          ORL    A,#30H        ;CONVERT LOWER DIGIT TO ASCII
          MOV    @R1,A         ;PUT INTO RESULT BUFFER
          INC    R1            ;POINT TO NEXT RESULT BUFFER LOCATION
          POP    ACC           ;GET BACK RESULT OF ADDITION
          SWAP   A             ;GET UPPER DIGIT
          ANL    A,#0FH
          ORL    A,#30H        ;CONVERT TO ASCII
          MOV    @R1,A         ;PUT INT RESULT BUFFER
          INC    R1            ;POINT TO NEXT RESULT BUFFER LOCATION
          DEC    R0            ;POINT TO NEXT TWO BCD DIGITS IN AUGEND
          DEC    R0            ;  ("DEC' DOESN'T AFFECT CARRY FLAG)
          DJNZ   R2,LOOP       ;LAST PAIR OF BCD DIGITS ADDED?
                               ;  NO—ADD TWO MORE
                               ;  YES—CHECK LAST DIGIT OF SUM
          JNC    CLRMSD        ; CARRY OUT OF 6 DIGIT ADDITION?
          MOV    @R,#31H       ;  YES-PUT AN ASCII "1" IN RESULT BUFFER
          SJMP   DONE
CLRMSD:   MOV    @R1,#30H      ;  NO-PUT AN ASCII "0" IN RESULT BUFFER
DONE:     POP    R0X           ; RESTORE PARAMETER POINTER
          SJMP   DONE
BUFFER:   DS     10H           ; BUFFER TO HOLD ASCII RESULT STRING
          END
```

Figure 2-6 Listing of the Demonstration Program

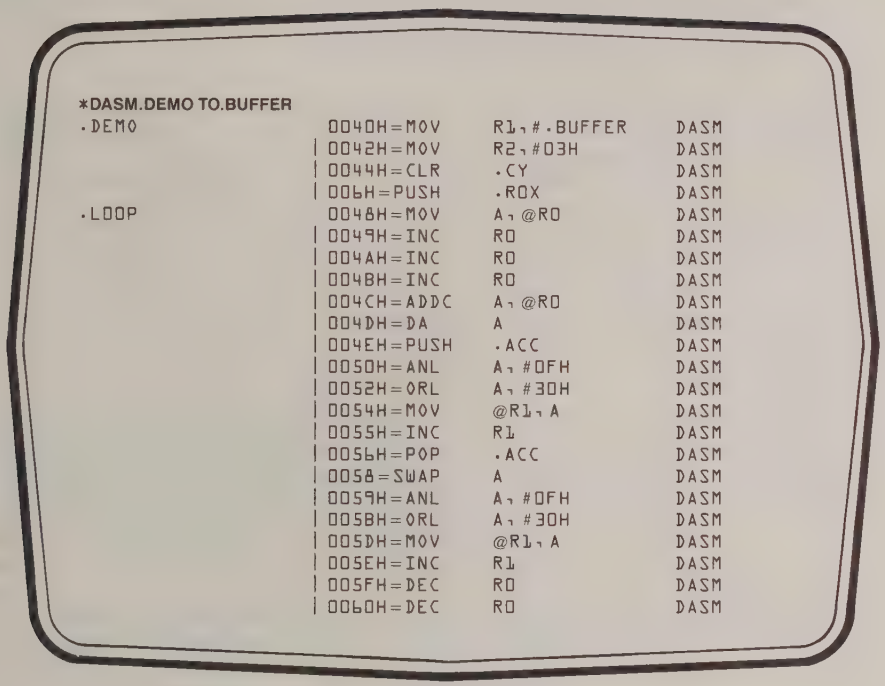
The BREAK (BR) command displays a table containing a list of all the breakpoints. The following example illustrates the BREAK command being entered and the address value breakpoint specified. The emulator software displays the breakpoint table showing one address breakpoint set from a previous operation. Breakpoint BR0 is set at address 0061H.

Key-in Sequence	Comments
BR RETURN	The BREAK command is entered and the emulator software displays the breakpoint table showing one address breakpoint from a previous operation. Breakpoint BR0 is set at address 0061H.



3. The DASM command disassembles the program and displays it on the screen. Display the demonstration program and verify that it was loaded by entering the command as follows:

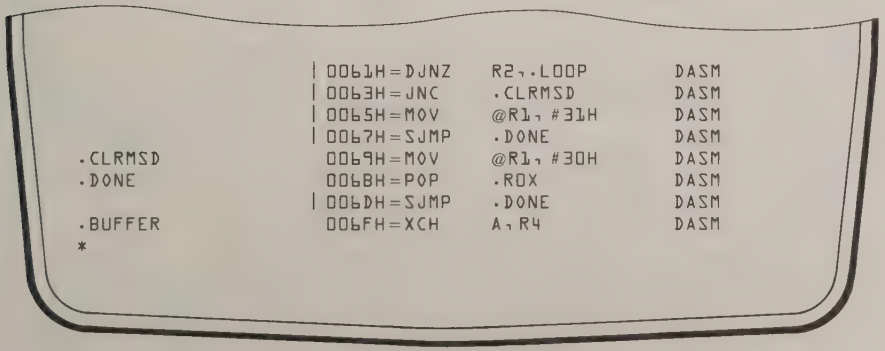
Key-in Sequence	Comments
DASM .DEMO TO .BUFFER RETURN	The DASM command disassembles the contents of program memory and produces the mnemonic equivalents of the instructions. In this example, memory from symbolic address .DEMO to .BUFFER is disassembled.



```
*DASM.DEMO TO.BUFFER
.DEMO          0040H=MOV    R1,#.BUFFER    DASM
               0042H=MOV    R2,#03H        DASM
               0044H=CLR    .CY            DASM
               006H=PUSH    .ROX           DASM
.LOOP          0048H=MOV    A,@R0          DASM
               0049H=INC    R0             DASM
               004AH=INC    R0             DASM
               004BH=INC    R0             DASM
               004CH=ADDC   A,@R0          DASM
               004DH=DA     A              DASM
               004EH=PUSH   .ACC           DASM
               0050H=ANL    A,#0FH         DASM
               0052H=ORL    A,#30H         DASM
               0054H=MOV    @R1,A          DASM
               0055H=INC    R1             DASM
               0056H=POP    .ACC           DASM
               0058=SWAP    A              DASM
               0059H=ANL    A,#0FH         DASM
               005BH=ORL    A,#30H         DASM
               005DH=MOV    @R1,A          DASM
               005EH=INC    R1             DASM
               005FH=DEC    R0             DASM
               0060H=DEC    R0             DASM
```

If information scrolls off of the screen, the following commands allow you to control the amount of data displayed on the screen.

Key-in Sequence	Comments
CTRL S	To halt scrolling of the screen display, press the CTRL key and the S key simultaneously.
CTRL Q	To continue, press the CTRL key and the Q key simultaneously.



```
               0061H=DJNZ   R2,.LOOP       DASM
               0063H=JNC    .CLRMSD        DASM
               0065H=MOV    @R1,#31H       DASM
               0067H=SJMP   .DONE          DASM
.CLRMSD        0069H=MOV    @R1,#30H       DASM
.DONE          006BH=POP    .ROX           DASM
               006DH=SJMP   .DONE          DASM
.BUFFER        006FH=XCH    A,R4           DASM
*
```

4. Enter the GO command to begin real-time emulation. The EMV-51A software begins emulation at the address specified with the command entry or at the current address in the 8051 program counter. Emulation continues at the full clock rate until a breakpoint is encountered.

5. When a breakpoint is encountered, the EMV-51A hardware stops emulation. At this point, instruct the emulator software to display the previous two instructions emulated and determine if the instructions are performing the desired operation.

6. By alternating between setting breakpoints and emulation of the program, testing and debugging of the prototype program can be done quickly.

7. At the end of the emulation session, the debugged code can be saved on an ISIS-PDS disk file, using the EMV-51A SAVE command.

Key-in Sequence	Comments
GO FROM .DEMO RETURN	The GO command emulates the user's program from an address entered with the command, or from the address contained in the program counter. The emulation continues until a breakpoint is reached. When a breakpoint is reached, the emulator software displays the cause of the breakpoint and the address where it occurred.
P RETURN	After an emulation is performed, enter the P command to display the last two instructions emulated. In this example, the instructions DEC and DJNZ are the last two instructions emulated with the GO command.
SAVE :F0:EMVDEM.V10 RETURN	Save the prototype program in a disk file prior to terminating the session. In this example, the program is saved in a file labeled EMVDEM.V10 on the disk in drive 0.

```
*GO FROM.DEMO
                    | 0061H=DJNZ   R2,.LOOP        GO-BREAK
*P
                    | 0060H=DEC    R0              DASM
                    | 0061H=DJNZ   R2,.LOOP        DASM
*SAV :F0:EMVDEM.V10
*
```

8. To return control to the ISIS-iPDS software, enter the EXIT command followed by a carriage return.

CHAPTER 3
TECHNICAL DESCRIPTION

Introduction

This chapter provides technical information for user-specified applications.

Controller

Figure 3-1 shows the location of the terminal posts and Table 3-1 describes these signals.

Table 3-1 User Test Signals

Test Signal	Description	Comments
CLK	Instruction fetch clock	CLK indicates the first fetch of the byte of an instruction from program memory (either the prototype's program memory or the EMV-51A's 8K RAM).
ALE	Address latch enable	The falling edge of ALE indicates that the 8051 microcontroller's address bus contains a valid address. The signal is high at the start of every bus cycle.
PSENE/	Program store enable	PSENE/ strobes data between the controller and the emulator. When PSENE/ is low, data passes from the controller to the emulator.
EMUL	Emulation	EMUL is high when the EMV-51A is emulating the user's program. It is high for one instruction prior to the start of program execution and stays high until the instruction following a breakpoint is fetched.
EXTBRK	External break input	The rising edge of EXTBRK from the target system stops the emulation currently in progress. The signal is terminated in a 220/330 ohm resistor network.
GND	Signal ground	The GND signal provides ground.

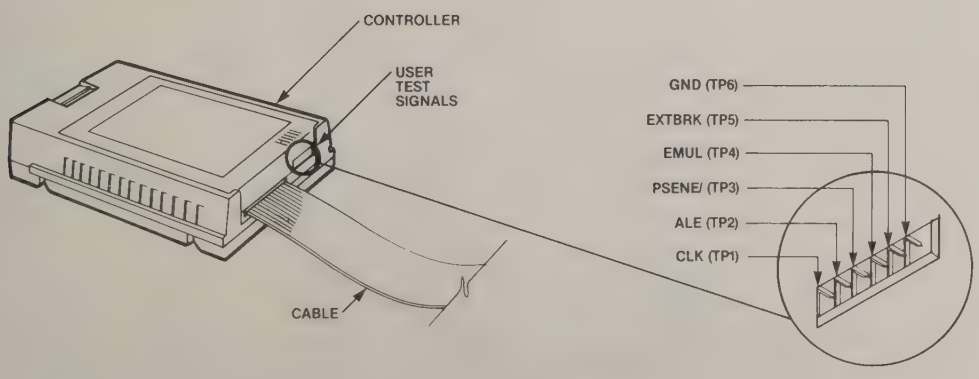


Figure 3-1 EMV-51A Controller's Terminal Post Location

When using a logic analyzer with the EMV-51A, use the EMUL signal as a triggering input to the logic analyzer. The logic analyzer feeds a signal to the EXTBRK input to halt emulation.

Emulator

The power for the emulator is supplied by the iPDS system or the user's system. The clock signal is supplied by a 12 MHz crystal located on the emulator circuit board or by the user's system. A power source and clock signals are selected via jumper connections.

Port 0 in Stand-Alone Mode

To read and write port 0 of the 8051 in stand-alone mode, the port must be pulled up by a resistor network, one resistor for each line on port 0. Figure 3-2 is a schematic drawing of the necessary connections.

Also, a connection must be made between pins E4, E5, and E6. A shorting plug connecting E4 and E5 along with a wire connection to E5 and E6 will suffice.

NOTE

Pin 40 in Figure 3-2 must be connected to ground.

Jumper Configurations

The EMV-51A emulator board contains five jumpers that select the supply voltage source, the clock source, the reset source, and internal or external program memory. These jumpers are listed in Table 3-2.

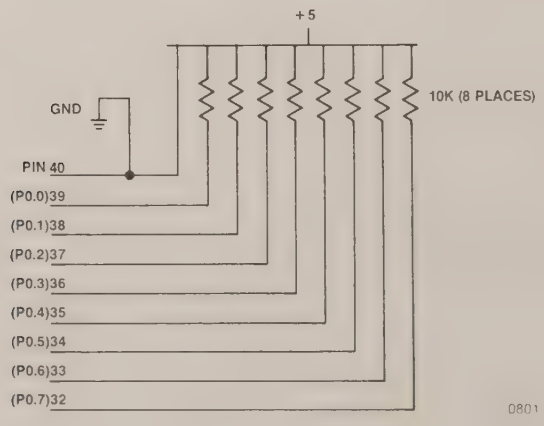


Figure 3-2 Port 0 Pullup Configuration

Table 3-2 EMV-51A Emulator Jumper Descriptions

Jumper Configuration	Description
E1-E2	EA/pin strapped for stand-alone mode
E2-E3	EA/state selected by user's prototype
E3-E4	Emulator powered by EMV-51A controller
E5-E6	Emulator powered by user's prototype
E7-E8	Reset provided by EMV-51A
E8-E9	Reset provided by user's prototype
E11-E12, E14-E15	Crystal provided by EMV-51A
E11-E10, E13-E14	Crystal provided by user's prototype

The reset jumper enables selection of the reset input to the emulator's 8051 bond-out. Either the user's system reset or a 10 uf reset capacitor on the processor module circuit board may be selected. The choice between internal emulator memory or external user memory is determined by the state of the external access (EA/) pin.

The following sections describe the various configurations for the four jumpers. Figure 3-3 shows the jumper locations on the EMV-51A emulator board.

Selecting the Program Memory. Use jumper configuration E1-E2 for stand-alone operation. Use jumper E2-E3 when the target system selects the state of the external access (EA/) pin. To use EMV-51A supplied code memory to debug code in a target system with EA/ strapped low, use jumper E1-E2.

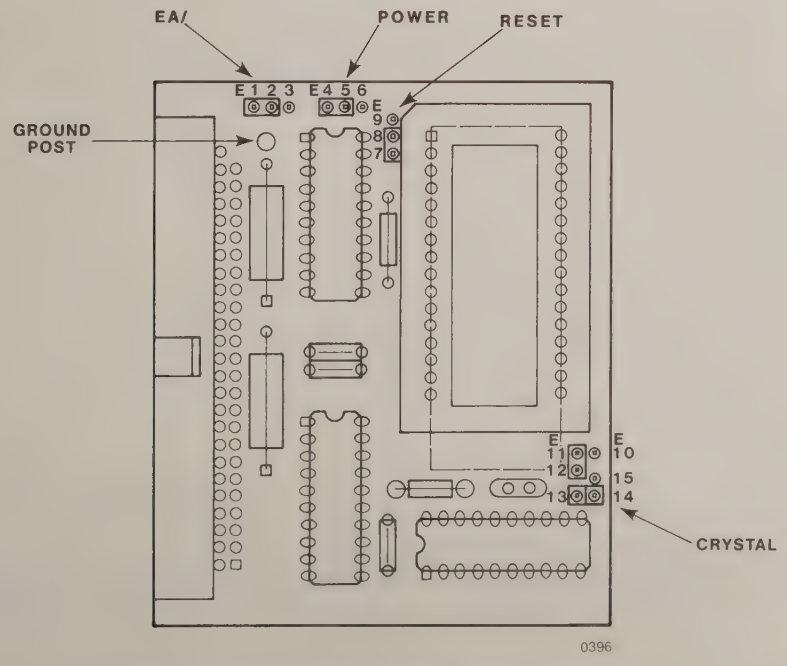


Figure 3-3 EMV-51A Emulator Jumper Locations

Selecting Supply Voltage. The jumper is moved to the E5-E6 position if supply voltage is provided by the target system. If the EMV-51A controller is providing the supply voltage, the jumper is moved to the E4-E5 position.

Selecting Oscillator Input. The jumpers are moved to the E10-E11 and E13-E14 positions if the clock is supplied by the target system. The jumpers are moved to the E11-E12 and E14-E15 positions if the clock is supplied by the emulator's on-board crystal.

Selecting Reset Input. If the reset capacitor on the emulator is to be connected to the 8051, move the jumper to the E7-E8 position.

If the target system is to supply the reset signal to the 8051, move the jumper to the E8-E9 position.

CHAPTER 4
USING THE EMV-51A

Introduction

This chapter presents an overview of the EMV-51A command categories. It describes how to enter commands and defines basic terms used throughout the manual. The following topics are covered:

- Command categories
- Prompts and messages
- Entering commands
- Entry editing and display control

Command Categories

The four categories of EMV-51A commands and the chapters in the manual describing each category are as follows:

Utility	Chapter 5
Display/Modify	Chapter 6
Emulation	Chapter 7
Advanced	Chapter 8

The utility commands are used to load user's programs from disk, establish symbolic names, save user's programs when the emulation session is completed, and operate on the emulator hardware. Also included in the utility commands section is the HELP command, which provides on-line information for all of the commands.

The display/modify commands are used to display or change register and memory contents. This category includes commands to assemble new instructions into code memory and to disassemble the contents of code memory.

The emulation commands are used to start and stop emulation of the user's program. This category includes commands to set up various emulation controls.

The advanced commands are used to build macro definitions and to set up conditional control structures. This category includes commands that define macros, load and save macro definitions, and assign special function keys to macros.

Entering Commands

After the emulator software has been loaded using the EMV51A command, the EMV-51A system is controlled by commands entered on the development system console or from command files.

Command Entry from the Console

An EMV-51A command is a sequence of one or more command words separated by spaces. The command line is limited to 122 characters. In most cases, the command words can be abbreviated to three characters. The command dictionary in Chapter 10 contains an alphabetical list of valid command words and their abbreviations.

The sequence of command words is terminated by a carriage return. With two exceptions, the carriage return causes the command to be interpreted, and if no syntax errors are detected, the command is executed. The exceptions are macro commands and continuation lines. Continuation lines are in the following section. Refer to Chapter 8 for details on macro commands.

Continuation Lines

To break a command line that exceeds the 122 character limit into two or more input lines, enter an ampersand (&) where a space would occur to request a continuation line. After the ampersand, the next carriage return is treated as an intermediate carriage return and nothing is executed until the final carriage return is encountered. Characters between the ampersand and the intermediate carriage return are ignored. The emulator software uses a double asterisk (**) prompt to indicate the beginning of a continuation line.

Comment Lines

Comments may be included at the end of the command line. Comment lines are preceded by a semicolon (;) and ended by a carriage return. The emulator software ignores all text between the semicolon and carriage return. An ampersand within a comment line does not act as a continuation character.

Entry Editing and Display Control

The emulator software recognizes control characters that permit editing of command line entries and control the display generated by command execution. These control characters are described in the following sections.

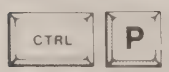
Command Line Editing

The following control characters are used to correct errors in command line entry. The control characters consist of one or two keyboard characters. If a character is preceded by the word CTRL in this manual, press the specified character while holding down the CTRL key on the iPDS keyboard. Otherwise, enter only the indicated key.

Aborts any operation that is executing or aborts any command being entered.

Signifies the end of the command line.

Deletes the last character typed. One character is deleted each time the key is pressed.



Gets the next keyboard entry and places the character in the command line buffer without interpreting it.

Echoes the command line being entered.

Deletes only the current line, not any previous lines. If several continuation lines have been entered, CTRL-X deletes only the current portion.

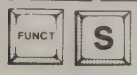
Deletes an entire command line, including all continuation lines.

Controlling the Display

The emulator software displays information on the screen by displaying a line, scrolling the displayed line upward one line, and displaying another line. This display speed can be controlled with the following characters.

Stops the display of new information to the screen.

Continues the display of new information to the screen.



Slows down or speeds up the screen display. Pressing FUNCT-S once slows the display down, while pressing it again speeds up the display.

Command Entry from Files

EMV-51A commands can be read in from an ISIS-PDS file if the command files follow the same command line conventions noted in the previous sections. The three types of files are as follows:

1. Previously defined command files containing valid EMV-51A commands. The command file is loaded and executed using the INCLUDE command (described in Chapter 8).

2. Macros, defined as a sequence of related commands, stored in a disk file. The macro definition is loaded and executed using the INCLUDE command (macros and the INCLUDE command are described in Chapter 8).

3. Previously defined files loaded and executed using the SUBMIT command. Refer to the *iPDS™ User's Guide* for additional information on the SUBMIT command.

Prompts and Messages

The EMV-51A software displays an asterisk (*) at the beginning of a line indicating readiness for command line entry and a special prompt (.*) during the macro definition (described in Chapter 8). Entering the WRITE command (described in Chapter 8) and an embedded RETURN causes the display of a double asterisk (**) prompt until a final RETURN is entered. For example, if you enter the following sequence, the first RETURN is embedded in the command line and the second RETURN terminates the command:

```
*WRITE 'THIS IS A [RETURN]
**TWO LINE MESSAGE' [RETURN]

THIS IS A
TWO LINE MESSAGE
*
```

Error or warning messages indicating the status of operations are displayed by the emulator software. These messages are only displayed when an error occurs. Appendix A contains a complete listing of the error and warning messages.

Abbreviated Commands

Many command can be abbreviated to fewest number of unambiguous letters. For example, BREAK may be abbreviated to BR, ENABLE may be abbreviated to ENA, and SAVE may be abbreviated to SAV. When commands can be abbreviated, the minimum letters required to enter the command will be underlined in this manual as follows:

BREAK

ENABLE

SAVE

CHAPTER 5
UTILITY COMMANDS

Introduction

The utility commands perform the emulator control functions for accessing the iPDS resources and display information about the emulator. This chapter describes the following utility commands:

• EMV-51A invocation	Initiates the emulator system from the ISIS-iPDS operating system.
• EXIT command	Terminates the emulation session and returns control of the console to the ISIS-iPDS operating system.
• HELP command	Displays brief explanations of the emulator commands.
• LOAD command	Loads the user program (and symbol table if desired) into the emulator's memory from the object file on disk.
• SAVE command	Copies the contents of the emulator code and symbol table memory to a disk file.
• LIST command	Copies the output from the emulation to a device other than the console.
• Symbolic reference commands	DEFINE, change, display, SYMBOLS, REMOVE, ENABLE/DISABLE SYMBOLIC.
• RESET command	Restores the emulator hardware to its initial state (as after invocation).
• SUFFIX command	Selects the implicit radix for numbers entered at the console.
• BASE command	Selects the default radix for certain kinds of numeric displays on the console.
• EVALUATE command	Displays the result of any expression in four radixes, as ASCII, and symbolically.

EMV-51A Invocation

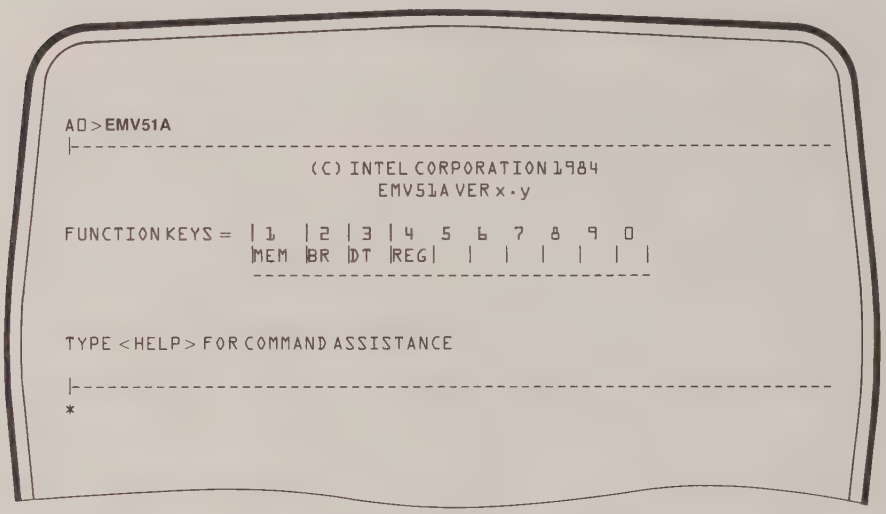
The emulator software is contained in an ISIS-iPDS file named EMV-51A. When this file is executed, the emulator hardware is initialized and the user controls the emulator through the console. The emulator software is invoked when the following steps are performed.

- Install emulator hardware following the procedure in Chapter 2.
- Install emulator software following the procedure in Chapter 2.
- Invoke the emulator software using the following command syntax:

Key-in Sequence	Comments
:F*n*:EMV51A	The ISIS-iPDS operating system loads and executes the EMV-51A software, which displays the sign-on message and the asterisk prompt shown in the screen display.

Where *n* is the number of the disk drive containing the EMV-51A software disk. (:F0: is the default and can be omitted from the command.) In the following screen display, the

iPDS prompt is shown as A0>. Once the EMV51A command is entered, the EMV-51A emulator software clears the iPDS sign-on message and provides its own sign-on message followed by the EMV-51A prompt, the asterisk (*).



EXIT Command

To end the emulator session and return control of the console to ISIS-iPDS, enter the following command:

Key-in Sequence	Comments
EXIT RETURN	The EXIT command halts all emulation, resets the emulator hardware, closes all files opened by the emulator, then returns control to ISIS-iPDS.

HELP Command

The HELP command displays information about the emulator commands and keywords. It operates in two modes: general help mode where a list of available commands and keywords is provided and specific mode where format and requirements about specific commands and keywords are displayed.

Help Information Screen Displays

The following screen displays show the two forms of the HELP command. The first form, without parameters, displays a list of all available commands and keywords.

```
*HELP
Help is available for the following commands and deinitions. Type HELP
followed by the full command name or <definition>.

utility        - BASE,DEFINE,DISABLE,ENABLE,EVALUATE,EXIT,LIST
                 LOAD,REMOVE,RESET,SAVE,SUFFIX,SYMBOL
display        - ASM,CBYTE,CDUMP,DASM,DBYTE,DDUMP,INTERRUPT
                 MEMORY,PBYTE,RBIT,RBYTE,REGISTER
emulation      - BREAK,DTRACE,GO,PREVIOUS,STEP
advanced       - COUNT,FUNCTION,DIR,IF,INCLUDE,PUT,REPEAT,WRITE
<definition>   - <ADDRESS>,<CHANGE>,<CPU$KEYWORD>,<DISPLAY>,<EXPR>,
                 <IDENTIFIER>,<INSTRUCTION>,<MACRO$DISPLAY>,
                 <MACRO$INVOCATION>,<NUMERIC$CONSTANT>,<PARTITION>,
                 <STRING>,<STRING$CONSTANT>,<SYMBOLIC$REF>,
                 <USER$SYMBOLS>

Keyboard-RUBOUT=delete char  ESC=cancel          CTRL R=echo line
Controls  CRTL X=delete line  CTRL P=literal char CTRL Z=delete line

Function Keys are invoked by simultaneously pressing FUNCTION
and a number.
      1 = MEMORY          3 = DTRACE     [5 to 9 and 0 are user defined]
      2 = BREAK           4 = REGISTER
```

The next example demonstrates HELP LOAD which responds with the information about the LOAD command. Typing in HELP followed by one or more commands or keywords produces information about each command or keyword.

```
*HELP LOAD
LOAD - Command to load object code and/or its symbols. The symbols are
added to the end of the EMV-51 symbol table.
   (1) To load the object code and its symbols, type:
       LOAD :Fn:<filename>
       where n is the drive number.
       (EX: LOAD :F2:MYPROG.OBJ)
   (2) To load only the ojbect code, type:
       LOAD :Fn:<ISIS filename>NOSYMBOL
       (EX: LOAD :F2:MYPROG.OBJ NOSYMBOL)
   (3) To load only the symbols, type:
       LOAD :Fn:<ISIS filename>NOCODE
       (EX: LOAD :F2:MYPROG.OBJ NOCODE)
NOTE: The LOAD command does not check for duplicate symbols. Thus, two
      loads of the same program will duplicate all symbols twice. Also,
      duplicate symbols can be accessed by entering the module name
      first.
      Example: (BYTE <.module.symbol><cr>
*
```

LOAD Command

The user program to be emulated typically resides in an object file generated by the ASM51 assembler. If the DEBUG assembler control has been used, the object file also contains the program's symbol table. See the *MCS® -51 Macro Assembly User's Guide* for information on using ASM51 and DEBUG control.

The LOAD command copies either hex or binary object code from a file into code memory, using the emulator memory map to identify the physical location of every address (see Chapter 6). Programs are loaded into the user-selected 4K or 8K bytes of EMV-51A program memory. When loading is complete, the program counter (PC) is set to the load address from the file (address of first executable instruction).

When the symbol table is included in the object file, the LOAD command optionally loads the program, the symbol table created with the assembler, or both the program and the symbol table. To load a file, enter the LOAD command as follows:

Key-in Sequence	Comments
LOAD EMVDEM.OBJ RETURN	This command loads EMVDEM.OBJ from a disk in drive number 0. Both the program and symbol table (if existent) will be loaded.
LOA EMVDEM.OBJ NOSYMBOL RETURN	This command loads the object code of EMVDEM.OBJ from a disk in drive number 0, but not the symbol table.
LOA EMVDEM.OBJ NOCODE RETURN	This command loads the symbol table from EMVDEM.OBJ from a disk in drive b number 0, but not the object code.

Loading Multiple Programs

The EMV-51A software permits loading of multiple programs into its program memory. The programs must be addressed so they do not overlap when loaded. This addressing is accomplished with the ORG statement during program assembly.

When loading multiple programs and symbol tables into the EMV-51A, each symbol table is appended to the end of the emulator software's internal symbol table with the program name preceding the first symbol. To load multiple programs, enter the following command:

Key-in Sequence	Comments
LOA EMVDEM.OBJ TEST.OBJ RETURN	This command loads the object code and symbol table for EMVDEM.OBJ and TEST.OBJ.

SAVE Command

The SAVE command writes the user's program to a disk file. The options available for the SAVE command make it possible for the user to specify whether the program, its symbol table, both the program and symbol table, or a range of project object code is written to the disk file.

Key-in Sequence	Comments
SAVE EMVDEM.SAV RETURN	Saves the program object code and symbol table in the file labeled EMVDEM.SAV.
SAV EMVDEM.SAV NOSYMBOL RETURN	Saves the program object code in the file labeled EMVDEM.SAV, but does not save the symbol table.
SAV EMVDEM.SAV NOCODE RETURN	Saves the program symbol table only. The object code is not saved.
SAV EMVDEM.SAV 10 TO 90 RETURN	Saves a range of program object code from address 10H through 90H and the symbol table in the file labeled EMVDEM.SAV.

LIST Command

In normal operation, the emulator software expects input from the keyboard and displays the results on the screen. Instructing the emulator software to put a copy of keyboard input and screen output in a separate file is accomplished by using the LIST command.

The LIST command instructs the emulator software to open a file to receive a copy of all information sent to the display screen or coming from the keyboard. This copy can be analyzed thoroughly after the emulation session to isolate errors. The following examples show how the LIST command is entered to open and close the list file.

Key-in Sequence	Comments
LIST :F1:EMVDEM.SMP RETURN	This command opens a list file named EMVDEM.SMP on drive 1. All subsequent keyboard and display information is copied to the file EMVDEM.SMP. Any information previously in the file specified with the LIST command is destroyed.
LIS :LP: RETURN	This command instructs the emulator to direct the copy to a line printer.
LIS :CO: RETURN	This command instructs the emulator to close the list file and display the information on the screen only. The list file now contains a complete copy of the emulation session.

If an emulation session is terminated prior to closing the LIST file, the emulator software closes the file before returning control to the ISIS-iPDS operating system.

The following screen display demonstrates loading a demonstration program, opening a LIST file, and closing the LIST file.

```
*LOAD EMVDEM.OBJ
*LIST :F0:SAVE.TXT
*SAVE :F0:EMV.PAR NOCODE
*LIST :C0:
```

Symbolic Reference Commands

A *symbol* is a name for a specific memory address or for a user-defined value. A *symbolic reference* in the emulator language is the name of a symbol preceded by a period (.). For example, if the user program contains a symbol START for code address 0100H, the symbolic reference .START would represent 0100H when it is used in a command such as PC=.START.

The emulator maintains two kinds of symbol tables, one for system-defined symbols and another for user-defined symbols.

System-Defined Symbols

The emulator system maintains tables of system symbols for addresses corresponding to the predefined symbols in the assembly language. The format of a system symbol in the emulator language is the name of the assembler symbol preceded by a period (.). For example, the assembler symbol SINT refers to the address of the serial port interrupt vector in code memory. In the emulator language, the corresponding system symbol is .SINT.

The system symbols are loaded with the emulator software at invocation. They are not part of the user symbol table, and are not affected by the user symbol table commands. The system symbols may not be changed or removed. Their numeric addresses are not displayed with the user table (SYMBOLS command), but can be displayed individually by entering the system symbol with its leading period, as shown in the following example. For on-line information on system-defined symbols, enter the command HELP SYSTEM$SYMBOLS.

Key-in Sequence	Comments
.SINT	This command instructs the emulator to display the numeric address for .SINT.

User-Defined Symbols

The emulator system maintains a symbol table for user-defined symbols. The user table contains the name of all user symbols in the order they were loaded or defined. The user table is initially empty.

The user table receives the assembler generated symbol table when the user program is loaded into the emulator using the LOAD command. When the DEBUG control is added to the assembler invocation, the assembler output file contains the user symbol in loadable format. The system loads the symbol table in the order the symbols appear in the program. The segment type (CSEG, DSEG, etc.) for user symbols is not loaded. User symbols in the emulator system are untyped. The system checks for and rejects system symbols in the user table.

NOTE

To use DEBUG control, use the P option, as in the command ASM51 EMVDEM.SRC P. This will instruct ASM51 to assemble the EMVDEM source file and create a symbol table. See Chapter 6 of the *MCS®-51 Macro Assembler User's Guide* for further information.)

DEFINE Command

The DEFINE command is used to assign expression values to symbol names. The expression can be a numeric value or an address. User defined symbol names must have a period as the first character. Once assigned, the symbol name rather than the expression can be entered, as shown in the following examples.

Key-in Sequence	Comments
DEFINE .START = 0100H RETURN	This command defines .START to address location 0100H.
DEFINE .LOOP = 0200H RETURN	This command defines .LOOP to address location 0200H.
DEF .BAKER = 2000T RETURN	Assigns the numeric decimal value 2000 to the symbol name .BAKER.
DEF .ABLE = 23T * 100T/110T RETURN	Assigns the result of the expression 23T * 100T/110T to the symbol name .ABLE.

The new symbol-name contains a maximum of 31 characters. The first character in the new symbol-name must be an alphabetic character, an at sign (@), or a question mark (?). The remaining characters can be alphanumeric characters, the at sign (@), the question mark (?), or the underscore (_).

Changing Symbol Values

The values assigned to symbol names can be changed by entering the symbol name followed by an expression or a value, as shown in the following examples.

Key-in Sequence	Comments
.ABLE = .ABLE + 1 RETURN	Increments the current contents of the symbol name .ABLE.
.BAKER = (.ABLE/256T) + 30H RETURN	Assigns the value of the expression (.ABLE/256T) + 30H to the symbol name .BAKER.

Displaying Symbol Names

To display a symbol name and its corresponding value, enter the symbol name preceded by a period.

Key-in Sequence	Comments
.ABLE RETURN	Display the value assigned to the symbol name .ABLE.
.BAKER RETURN	Display the value assigned to the symbol name .BAKER.

Displaying Multiply Defined Names

When several program modules are loaded into EMV-51A program memory, a symbol name might be defined more than once. When this happens, a specific definition can be displayed by preceding the name with the module name that contains the desired definition.

Key-in Sequence	Comments
.EMVDEM.ABLE RETURN	Displays the value assigned to the symbol name .ABLE used in the program EMVDEM.
.EMVTWO.ABLE RETURN	Displays the value assigned to the symbol name .ABLE used in the program EMVTWO.

SYMBOLS Command

The SYMBOLS command displays all user symbol names currently residing in memory and their corresponding address values. The format of the command is as follows:

Key-in Sequence	Comments
SYMBOLS RETURN	This command displays all user-defined symbols and their corresponding address values.

REMOVE Command

The REMOVE command deletes one or more user-defined symbol names from the emulator software's symbol table, as shown in the following examples.

Key-in Sequence	Comments
REMOVE .BAKER RETURN	Removes the user defined symbolic name .BAKER from the symbol table.
REM .APPLE, .BAKER RETURN	Removes the user defined symbolic names .APPLE and .BAKER from the symbol table.
REM SYM RETURN	Removes all user defined symbol names from the symbol table.

Enabling/Disabling Symbol Display

The emulator software commands display results as either symbolic names or numeric data. The commands look at an internal software switch before displaying symbolic names. This switch is controlled with the ENABLE and DISABLE commands. The default condition is symbols enabled.

ENABLE Symbolic Expansion Command

The ENABLE command sets the internal switch permitting the display of symbolic names rather than numeric addresses. When the software finds a symbol in the symbol table that

matches the numeric data, it displays the symbol rather than the data. The ENABLE SYMBOLIC command affects the symbolic display format when using the DASM and PC commands. If symbolic display is enabled and a match is found, DASM displays opcode addresses as symbolic names.

In the following example, DASM is used to display the partition of memory from 40H to 45H. Note that the symbolic name is displayed as .CY.

```
*ENABLE SYMBOLIC
*DASM 40H TO 45H
.DEMO
                0040H=MOV     R1,#.BUFFER     DASM
              | 0042H=MOV     R2.#03H         DASM
              | 0044H=CLR     .CY             DASM
```

In the following example, the symbolic name .EMVDEM and its memory location are displayed; then PC is used to display the symbolic name and offset of the address after an emulation break.

```
*ENABLE SYMBOLIC
*.EMVDEM
.EMVDEM=0200H
*PC
PC=.EMVDEM+0133H
```

DISABLE Symbolic Expansion Command

The DISABLE command resets the internal switch, forcing the software to use numeric addresses. In the following example, DASM is used to display the partition of memory from 40H to 45H. Note that the numeric value for the symbol name .CY is displayed as D7H.

```
*DISABLE SYMBOLIC
*DASM 40H TO 45H
                0040H=MOV     R1,#.BUFFER     DASM
              | 0042H=MOV     R2.#03H         DASM
              | 0044H=CLR     D7H             DASM
```

If symbolic display is disabled, PC will not display an offset; it will only display the current memory address, as shown in the following screen display.

```
*DISABLE SYMBOLIC
*PC
PC=0333H
```

RESET Command

The RESET command initializes the breakpoints and *tracepoints* to the off state, puts the 8051 microcontroller in its reset state, and issues the asterisk prompt for additional commands.

Key-in Sequence	Comments
RESET RETURN	This command reinitializes the EMV-51A hardware and sets the DTRACE and BREAK commands to their default states.

The SUFFIX and BASE Commands

The EMV-51A software accepts numeric values entered from the keyboard and displays numeric values on the screen in one of four number bases: hexadecimal (H), decimal (T), binary (Y), or octal (Q). The desired number base is specified for input and display with two commands, SUFFIX and BASE.

Any number entered without an explicit suffix is interpreted using the current suffix. An explicit suffix overrides the suffix set by the SUFFIX command.

The number base for input via keyboard entry is specified with the SUFFIX command and the number base for display output is specified with the BASE command.

The SUFFIX command displays the current keyboard entry number base or changes the current number base to the specified one, as follows:

Key-in Sequence	Comments
SUFFIX RETURN	Displays the current keyboard entry number base.
SUFFIX=T RETURN	Changes the current keyboard entry number base to decimal. See the following screen display.

```
*SUFFX
H
SUFFIX=T
*
```

The BASE command displays te current display number base or changes the current number base to the specified one, as shown in the following example.

Key-in Sequence	Comments
BASE RETURN	Displays the current display number base.
BASE=Q RETURN	Changes the current display number base to octal. (See the following screen display.)

```
*BASE
H
BASE=Q
*
```

The keyboard entry and display number bases may differ, as the previous examples illustrate. Generally, the display is more readable when both bases are identical.

EVALUATE Command

The EVALUATE command displays the value of a previously defined symbol name or calculates the value of an expression involving symbols and mathematical operators. In either case, the value is displayed in hexadecimal, decimal, octal, ASCII, and as an offset from the nearest symbol name that is less than the value, as shown in the following examples.

Key-in Sequence	Comments
EVALUATE .LOOP RETURN	Display the numeric value or address currently assigned to .LOOP.
EVALUATE 100*23+4 RETURN	Calculate the value of the expression and display the result.

The following screen display illustrates the results from the previous examples.

```
*EVA .LOOP
1001000Y 1100 72T 48H 'H' .LOOP
*EVA 100*23+4
1000110000001000Y 21404Q 8964T 2304H '#' .EMVDEM+2104H
```

CHAPTER 6
DISPLAY/MODIFY COMMANDS

Introduction

The display/modify commands include register and memory handling commands. This chapter describes what the commands are, where they are used, and provides examples of how they are used.

Using Register Commands

The MCS-51 family of microcontrollers contain temporary data storage locations, called registers. A register is accessed by entering the keyword associated with that register. Table 6-1 lists the registers and the associated keyword names.

Table 6-1 Register Keyword Names

Keyword	Function
PC	Program Counter
SP	Stack Pointer
DPTR	Data Pointer
ACC	Accumulator
B	Multiplication Register
TM0	Counter/Timer 0
TM1	Counter/Timer 1
PSW	Program Status Word
RBS	Register Bank Select
R0	General Purpose Register 0
R1	General Purpose Register 1
R2	General Purpose Register 2
R3	General Purpose Register 3
R4	General Purpose Register 4
R5	General Purpose Register 5
R6	General Purpose Register 6
R7	General Purpose Register 7

Displaying Register Contents

The emulator software displays the contents of a register when the keyword associated with the particular register is entered, as shown in the following examples.

Key-in Sequence	Comments
PC RETURN	Displays the current address value in the program counter.
SP RETURN	Displays the current 8-bit address value in the stack pointer.
DPTR RETURN	Displays the current address value in the data pointer.

Modifying Register Contents

To modify the contents of a register, enter the register keyword followed by an expression. The following examples illustrate several types of expressions.

Key-in Sequence	Comments
ACC = 88H RETURN	Changes the contents of the accumulator to 88 hexadecimal.
R0 = .BCC + 22H*2 RETURN	Changes the contents of general purpose register R0 to the value of the expression, .BCC + 22H*2.
DPTR = 4444T RETURN	Changes the contents of the data pointer to 4444 decimal.

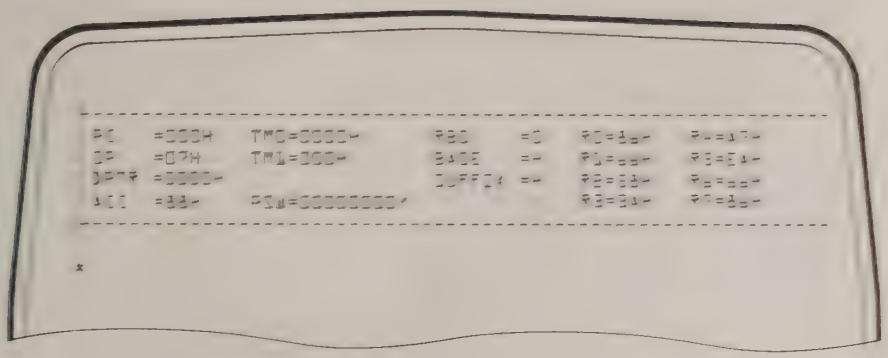
The REGISTER Command

The REGISTER command displays all register keyword contents on the screen. The contents of the registers are displayed in the current number base unless the number base is binary. If the current number base is binary, the contents of the registers are displayed in hexadecimal.

The display remains on the screen until a non-register keyword (i.e., HELP) is entered. Thus, several registers can be changed and the display updated to reflect the new values. If the BASE is changed, all register contents change to reflect the new base. All register contents are entered according to the current number suffix. The following examples show how to use the REGISTER command.

Key-in Sequence	Comments
REGISTER RETURN	The emulator software displays a table containing the 8051 registers. The command can be abbreviated to REG or R.
ACC = 88 RETURN	This command entry changes the 8051 accumulator to 88 hexadecimal since the suffix is currently H. The display will be rewritten to reflect the new value for the accumulator. (See the following screen display.)

```
*R
-------------------------------------------------------------------
 PC    =0000-    TM0=0000H       PBC     =0   P0=8e-   P4=47-
 SP    =07-      TM1=000H        BASE    =H   P1=ee-   P5=2AH
 DPTR  =0000-                    SUFFIX  =H   P2=88-   P6=ee-
 ACC   =00-      PSW=00000000Y                P3=3A-   P7=8e-
-------------------------------------------------------------------

*ACC=88
----
```



Special Function Key 4

Rather than enter the command REGISTER, function key 4 returns the same display. To do this, press the numeral key while holding down on the FUNCT key. For example:

Key-in Sequence	Comments
FUNCT 4	The emulator software displays a table containing the 8051 registers. Entering FUNCT 4 produces the same display that is produced when entering the REGISTER command, or either of its abbreviations (REG or R).

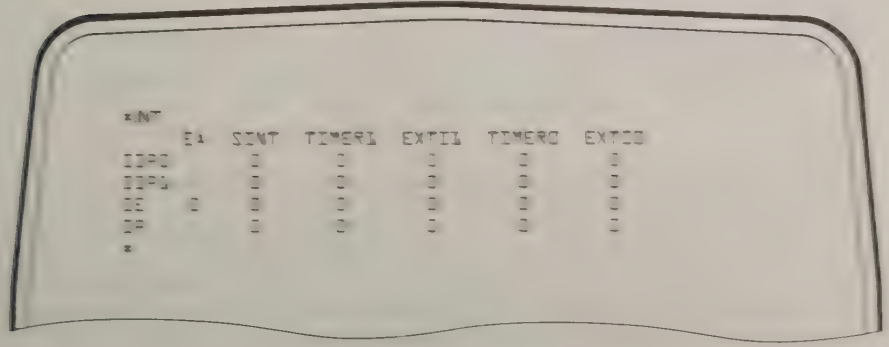
Displaying INTERRUPT Register Contents

The 8051 provides five interrupt sources, which are described in detail in the *Microcontroller Handbook*. The EMV-51A emulator software includes the INTERRUPT command that displays the interrupt register contents.

INTERRUPT Command

The INTERRUPT command displays the interrupt enable (IE), interrupt priority (IP) registers, and the status of interrupts in progress for priority 0 (IIP0) and priority 1 (IIP1). The command is entered as follows:

Key-in Sequence	Comments
INTERRUPT RETURN	This command displays a table showing the status of all 8051 interrupts. (See the following screen display.)



The mnemonics in the previous screen display are abbreviations for the following descriptions:

Mnemonic	Description
EA	Enable all interrupts (IE register only)
SINT	Serial port interrupt
TIMER1	Timer 1 interrupt
EXTI1	External interrupt 1
TIMER0	Timer 0 interrupt
EXTI0	External interrupt 0

Memory Commands

The MCS-51 microcontroller family contains separate program, data, and special function register memory areas. To access a memory area, enter the keyword associated with it. This keyword also serves as the command for modifying and displaying that memory area. Table 6-2 lists the memory areas and their keyword names, while Chapter 7 contains a breakdown of the contents of each memory area. The *Microcontrollers Handbook*, order number 210918, contains additional information on the architecture of the microcontrollers.

Table 6-2 Memory Areas and Keyword Names

Keyword	Memory Area
CBYTE	Program memory (0-4K internal/ 0K-8K internal)
DBYTE	Internal 128 byte data memory
PBYTE	External 64K byte data memory
RBYTE	Internal special register memory (ports, counter)
RBIT	Internal bit-addressable memory
CDUMP	Formatted display of CBYTE memory
DDUMP	Formatted display of external data memory

Displaying Memory Contents

The contents of a memory area are displayed when the keyword name and an address within that area are entered. The address consists of single or multiple locations. Following are several examples of memory display commands.

Key-in Sequence	Comments
CBYTE 03FFH RETURN	Displays the contents of a single memory location.
DBYTE 0 TO 20H RETURN	Displays the contents of a range of internal data memory locations from 0 through 20H.
PBYTE 37H LEN 9 RETURN	Displays the contents of external data memory locations from 37H to 3FH.

Optionally, the contents of program or external data memory can be displayed in both ASCII and hex numeric format, as shown in the following two examples.

Key-in Sequence	Comments
CDUMP 100H TO FFFH RETURN	Displays the contents of a range of program memory locations as ASCII characters and hexadecimal numbers.
DDUMP 0 TO 100H RETURN	Displays the contents of a range of external data memory locations as ASCII characters and hexadecimal numbers.

NOTE

Code memory from 0 to 0FFFH may be internal or external to the EMV-51A emulator when it is plugged into a user prototype system. If the emulator is part of a user prototype system, the external access (EA/) input to the emulator determines where the memory is located: EA/ = 1 means the memory from 0H to 0FFFH is internal to the EMV-51A emulator, while EA/ = 0 means the memory is external to the EMV-51A emulator. The memory data displayed by CBYTE or CDUMP is determined by the level of the EA/ pin. The EA/ pin can be strapped high or the user prototype system can determine the state of the EA/ pin.

Modifying Memory Contents

The contents of a memory location can be changed by entering the memory keyword, address, and an expression, as shown in the following command examples.

Key-in Sequence	Comments
CBYTE 03FFH = 33H RETURN	Changes the current contents of program memory location 03FFH to 33H in EMV-51A program memory.
PBYTE 03FFH = 55H RETURN	Changes the contents of external data memory location 03FFH to 55H.
DBYTE 0 = 2*3 + .ABLE RETURN	Changes the contents of data memory location 0 to the result of the expression 2*3 + .ABLE.

NOTE

CBYTE modifies only the EMV-51A program memory, which is located from 0 to 0FFFH. If the EMV-51A emulator is plugged into a user prototype system with the external memory at these addresses (EA/ asserted), the memory may be examined, but not modified.

Displaying Memory Commands

There are three ways to display a table containing memory keywords and their format requirements: entering the MEMORY command, the letter M, or the combination of the FUNCT key and the number 1, as shown in the following examples.

Key-in Sequence	Comments
MEMORY RETURN	Displays all memory keywords and their format requirements.
M RETURN	
FUNCT 1	

The table display and the memory command CBYTE are shown in the following screen display. In this example, the CBYTE command displays the contents of a specified section of internal program memory. The contents are displayed in the current base, which is hexadecimal. If an address is assigned to a symbol name and the name is in the symbol table, the EMV-51A software displays the symbol name followed by the address.

```
*M
|-------------------------------------------------------------
|   MEMORY COMMANDS
|-------------------------------------------------------------
| CBYTE (code memory  )|           | TO Location  | =value
| DBYTE (data memory  )|           |              |
| RBYTE (registers    )| Location  | LENGTH n
| RBIT  (bit flags    )|           |
| PBYTE (ext.data     )|           |
|-------------------------------------------------------------
| CDUMP (code dump)   | Location TO Location
| DDUMP (data dump)   |
|-------------------------------------------------------------
*CBYTE 40H LEN 69H
.DEMO
0040H=79H 6FH 7AH 03H C2H D7H C0H 00H
.LOOP
0048H=E6H 08H 08H 08H 36H D4H C0H E0H 54H 0FH 44H 30H F7H 09H D0H E0H
0058H=C4H 54H 0FH 44H 30H F7H 09H 18H 18H DAH E5H 50H 04H 77H 31H 80H
0068H=02H
.CLRMSD
0069H=77H 30H
*
```

Assembly/Disassembly Commands

The EMV-51A software contains two commands, ASM and DASM, which assemble program instructions in program memory and disassemble program object code into mnemonic equivalents. These commands are covered in the following sections.

The ASM Command

The ASM command performs three functions:

- Displays the current contents of the assembly location counter.
- Changes the current contents of the assembly location counter.
- Assembles one instruction into program memory.

The emulator software maintains an assembly location counter that indicates the next available location in program memory where new instructions can be placed. The current contents of the assembly location counter can be displayed by entering the following command.

Key-in Sequence	Comments
ASM RETURN	Displays the current contents of the assembly location counter.

New instructions are also assembled into program memory with the ASM command. The new instructions are stored in memory at the current location pointed to by the assembly location counter. The assembly location counter is incremented to the next available location and the new location is displayed, as shown in the following examples.

Key-in Sequence	Comments
ASM ORG 03FFH RETURN	Changes the contents of the assembly location counter to 03FFH.
ASM MOV A,R0 RETURN	Assembles the instruction MOV A,R0 into program memory at the location contained in the assembly location counter.

After a new instruction is assembled into program memory, the ASM command updates the assembly location counter and displays the new value.

NOTE

The ASM command only operates between memory addresses 0 and 0FFFH, inclusive, in EMV-51A internal program memory.

The DASM Command

The DASM command disassembles user programs, producing the mnemonic equivalents of the machine language instructions. The command disassembles a single instruction or a range of instructions, as shown in the following examples.

Key-in Sequence	Comments
DASM 100H TO 150H RETURN	Disassembles the contents of memory from address 100H through 150H, producing the mnemonic instruction equivalents.
DASM .ABLE TO .BAKER RETURN	Disassembles the contents of memory from symbolic address .ABLE through symbolic address .BAKER, producing the mnemonic instruction equivalents.

Using DISPLAY/MODIFY Commands

In Chapter 2, the demonstration program, EMVDEM.OBJ, was described (see Figure 2-6). If this program has been loaded, it can be assembled and disassembled as shown in the following examples.

Key-in Sequence	Comments
ASM RETURN	Displays the current value of the assembly location counter (ALC), which points to the place in program memory where the next instruction is assembled.
ASM ORG.BUFFER + 10 RETURN	Sets the ALC to a new value (by using the ORG command). The emulator software displays the new location immediately following the command.
DASM 40H TO 50H RETURN	Disassembles the contents of memory from addresses 40H to 50H and produces the mnemonic equivalents of the instruction.

The following screen display illustrates the commands entered by the user and the response of the emulator software.

```
*ASM
0000H
*ASM ORG.BUFFER + 10
007FH
*DASM 40H TO 50H
.DEM0                  0040H=JBC   6CH,0082H      DASM
                     | 0043H=ANL   A,R5           DASM
                     | 0044H=DJNZ  R1,FFD9H       DASM
                     | 0046H=RRC   A              DASM
                     | 0047H=ADDC  A,R6           DASM
                       0048H=MOV   .IE,R2         DASM
                     | 004AH=SUBB  A,R4           DASM
                     | 004BH=RLC   A              DASM
                     | 004CH=ORL   A,R3           DASM
                     | 004DH=SUBB  A,R2           DASM
                     | 004EH=DEC   @R1            DASM
                     | 004FH=JNB   DAH,004CH      DASM
```

CHAPTER 7
USING EMULATION COMMANDS

Introduction

This chapter describes the breakpoint, software trace, and emulation command groups. The following sections of this chapter describe the commands included in each group, discuss the commands use, and give command entry examples. The following topics are discussed in this chapter:

- Setting breakpoints
- Using breakpoint commands
- Using software trace commands
- Using program emulation commands
- Using emulation controls

Setting Breakpoints

A breakpoint functions as a switch that stops program execution after a specified event occurs. By using breakpoints, interactive control of program operation is possible.

When a breakpoint is encountered during program execution, the emulator stops program execution. Register and memory contents can then be selectively displayed and modified.

The emulator software supports five types of breakpoints: address, branch, range, arming, and register value. These breakpoint types are described in the following sections.

Address Breakpoints

The address breakpoint (BR*n*, where $n = 0$ to 3) stops program execution if control passes to a specified address. The address must be the first byte of an instruction; otherwise, the breakpoint is ignored and program execution continues. Up to four address breakpoints can be specified, as shown in the following examples.

Key-in Sequence	Comments
BR0 = 100H RETURN	Sets address breakpoint 0 for address 100H.
BR1 = .ABLE RETURN	Sets address breakpoint 1 for the address specified by the symbol name .ABLE.
BR2 = .ABLE + 5 RETURN	Sets address breakpoint 2 for the address located 5 bytes after the address specified by the symbol name .ABLE.
BR3 = 100H + .BAKER*2 RETURN	Sets address breakpoint 3 for the address specified by the expression 100H + .BAKER*2.

Branch Breakpoints

A branch breakpoint (BRB) stops program execution when a jump instruction is executed; for example, when a jump, call, return, or return from interrupt instruction is executed. The branch breakpoint is only used in the real time (or GO) emulation mode. Branch breakpoint control can be enabled or disabled by specifiying the breakpoint ON or OFF. The two possible command entries are as follows:

Key-in Sequence	Comments
BRB = ON RETURN	Enables the branch breakpoint control.
BRB = OFF RETURN	Disables the branch breakpoint control.

Range Breakpoints

A range breakpoint (BRR) stops program execution when program control passes to an address within a specified range of addresses. The beginning address must be the first byte of an instruction; otherwise, the breakpoints at these addresses are ignored and program execution continues. The following examples illustrate the range breakpoint command.

Key-in Sequence	Comments
BRR = 100H TO 120H RETURN	Sets a range breakpoint for the address range 100H to 120H.
BRR = .ABLE TO .ABLE + 100H RETURN	Sets a range breakpoint for the address range specified by .ABLE as the lower limit and .ABLE + 100H as the upper limit.

Register Value Breakpoints

The register value breakpoint (BV) stops program execution if a register contains a specified value. The register value breakpoint is only used in the single step (STEP) emulation mode. The user can specify one of the general purpose registers (R0 through R7) or the accumulator (ACC). The specific bank of general purpose registers depends on the contents of register bank select (RBS), which specifies register banks 0, 1, 2, or 3. The following examples illustrate how to specify a value for a register or the accumulator.

Key-in Sequence	Comments
BV = R0 44H RETURN	Stops emulation when general purpose register R0 contains the value 44H.
BV = ACC .ALPHA RETURN	Stops emulation when the accumulator contains the least 8 bits of the value represented by the symbol .ALPHA.

Clearing Breakpoints

The breakpoint clear (BC) command disables all breakpoints, as shown in the following example.

Key-in Sequence	Comments
BC RETURN	Disables all set breakpoints.

Breakpoint Restrictions

Up to four address breakpoints can be set at any given time. The range breakpoint (BRR) counts as three active address breakpoints. Thus, one range break and one address break can be specified, or up to four address breaks can be specified at once.

When two or more address breakpoints and a range breakpoint are specified, the emulator software displays the following error message:

TOO MANY BREAKS

The software does not recognize a range breakpoint when two or more address breakpoints are enabled. To set a range breakpoint in this case, clear one or more of the address breakpoints and then set the range breakpoint.

NOTE

When one range breakpoint and one address breakpoint are set and another address breakpoint is added, the range breakpoint is destroyed.

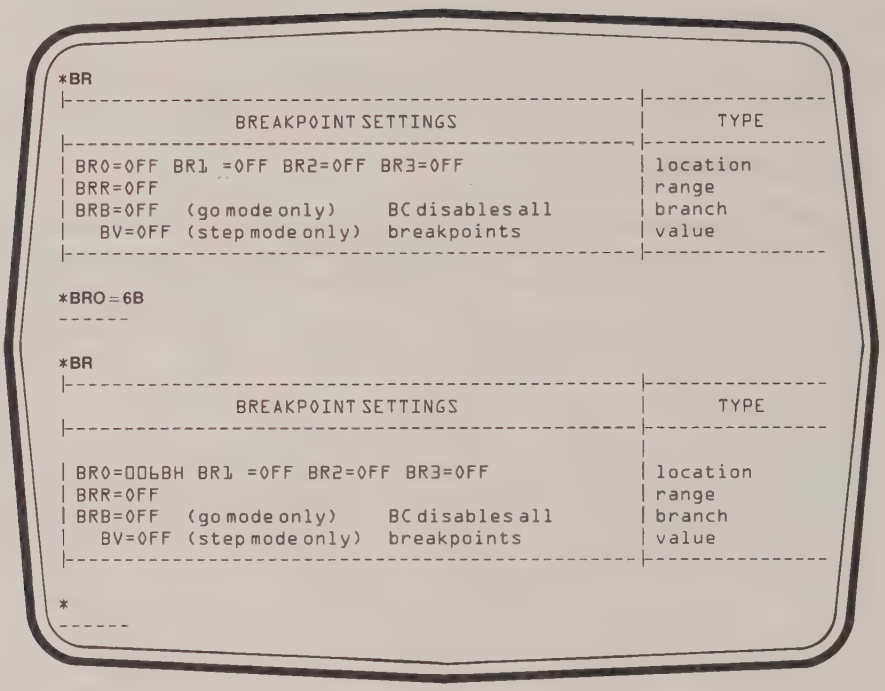
Using the BREAK Command

Breakpoint commands can be entered individually with the breakpoint commands previously described, or from within a breakpoint table. This breakpoint table is displayed by entering the BREAK command.

The BREAK command displays a table containing all breakpoints, their entry requirements, and the current state of each (see the following command examples). When a breakpoint command is entered, the displayed table is updated to reflect the new entry. The breakpoint display remains on the screen until a non-breakpoint command is entered.

Key-in Sequence	Comments
BREAK RETURN	Displays a table containing all the breakpoint commands.
BR RETURN	
BR0=6B RETURN	BR0 is enabled with the BREAK table.

The following display occurs when entering breakpoint commands.



Special Function Key 2

Holding the function key and pressing the number 2 also displays the BREAK table, as shown in the following example.

Key-in Sequence	Comments
FUNCT 2	Displays all breakpoint keywords and their contents on the screen.

Using the Trace Display

During STEP emulation of the user program, up to four lines of emulation information can be automatically displayed after each instruction is emulated. This information, called trace information, consists of disassembled instructions, register and memory contents, and status information.

The automatic display of the emulation information can be enabled or disabled depending on program execution addresses or register contents: The emulator software retains the last state of each trace display command. If a trace command is disabled, the command must be re-enabled before it can be used to display trace information. This section describes how to use the following commands:

- TD command
- TR command
- TB*n* command
- TS command
- TR*n* command
- TV command

The TD Command

The instruction trace (TD) command enables or disables the display of the disassembled instruction at a breakpoint or during STEP emulation mode. If enabled, the instruction at the breakpoint is disassembled and displayed, along with the cause of the breakpoint, on one line of the display. The possible command entries are as follows.

Key-in Sequence	Comments
TD = ON RETURN	Enables the display of disassembled instructions and the cause of the breakpoint.
TD = OFF RETURN	Disables the display of disassembled instructions.

The TR Command

The register display (TR) command enables or disables the display of general purpose registers R0 through R7, the accumulator (ACC) at a breakpoint or during STEP emulation mode, and the program status word (PSW). If enabled, the registers are displayed on one line of the screen. The two possible command entries are as follows:

Key-in Sequence	Comments
TR = ON RETURN	Enables the display of general purpose registers R0 through R7, the accumulator (ACC), and the PSW.
TR = OFF RETURN	Disables the display of general purpose registers R0 through R7, the accumulator (ACC), and the PSW.

The TB*n* Command

The trace bit (TB*n*) command enables or disables the display of up to four one-bit memory locations within the direct bit-addressable memory space when emulation halts, where *n* specifies one of four trace bit commands. Bit-addressable memory contains 128 user-defined software flags located between internal data memory addresses 20H through 2FH as well as several bit locations within several special function registers. Chapter 9 contains a memory map showing the physical layout of the bit-addressable memory. Following are two command entry examples.

Key-in Sequence	Comments
TB0 = 4H RETURN	Displays the contents of the fifth bit in bit-addressable memory as bit display TB0 when a breakpoint occurs or during STEP emulation.
TB2 = OFF RETURN	Disables the display of bit display TB2.

The TS Command

The trace status display (TS) command enables or disables the display of the program status word (PSW) register when a breakpoint occurs. The register contains seven 1-bit flags that indicate the current state of the following items:

- Carry flag
- Auxiliary carry flag
- User flag 0
- Register bank select RS0 and RS1
- Overflow flag
- Parity flag
- UTL (a utility flag that is reserved and not accessible to the user)

The two status display command entries are as follows:

Key-in Sequence	Comments
TS = ON RETURN	Enables the display of the program status word register (PSW) on one line of the screen display.
TS = OFF RETURN	Disables the display of the PSW.

Controlling the Trace Display

The commands previously described enable specific types of information to be displayed as part of the instruction trace display. The instruction trace display can further be controlled by actions occurring within the user's program.

The emulator contains two commands that enable or disable the instruction trace displays depending on the contents of an 8051 register or when program control passes to a specific address. These commands are covered in the following sections.

The TR*n* Command

The TR*n* command enables or disables the trace display (as specifi with the TD, TR*n*, TB*n*, and TS commands) depending on program control passing to a specified memory address. Up to four program locations can be specified that enable or disable the instruction trace display. These locations must be the starting byte of an instruction. The following example demonstrates how to use this command.

Key-in Sequence	Comments
TR0 = 100H ON RETURN	Enables the trace display if the instruction at address 100H also has a breakpoint set.
TR1 = 110H OFF RETURN	Disables the trace display at address 110H.

In this example, the trace information (specified with the trace display commands) is enabled if the instruction at address 100H is also the location of a breakpoint. In STEP mode, the trace information is displayed for each instruction executed until program control reaches address 110H. In GO emulation, the only instructions displayed are those that also have breakpoints set. At address 110H, the trace display is disabled. Once the trace display is disabled, no further trace information is displayed until the display is enabled again.

The TV Command

The TV command enables or disables the trace display (as specified by the TD, TR*n*, TB*n*, and the TS commands) depending on the contents of a specified register. Any of the registers R0 through R7 or the accumulator ACC can be specified. The following example demonstrates using this command.

Key-in Sequence	Comments
TV = R0 44H ON RETURN	Enables the trace display if general purpose register R0 contains 44H.
TV = ACC 10H OFF RETURN	Disables the trace display if the accumulator contains 10H.

In this example, the trace information is displayed if general purpose register R0 contains the value 44H. The trace information is displayed for each instruction executed until the accumulator contains the value 10H. At this point, the trace display is disabled. Once the trace display is disabled, no further trace information is displayed until the display is enabled again.

NOTE

If the software trace display is turned off with either the TR*n* or TV commands, the display remains off permanently until one o tommds which is set turn on tdisplay exected, or until the EMV-51A software is reset.

The DTRACE Command

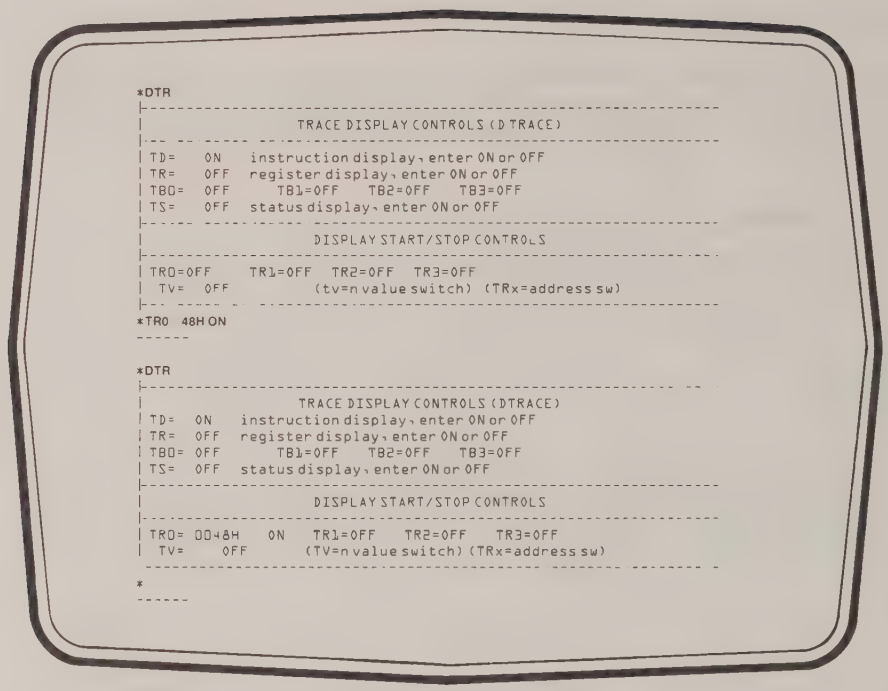
The software trace (DTRACE) command displays a table containing all the trace display controls. Thus, the trace control commands can be entered individually or interactively from the DTRACE table. As the instruction trace commands are entered, the table is updated to reflect the new entries. The DTRACE display remains on the screen until a non-trace display command is entered. The DTRACE table is then replaced by the results of the new command. The following commands will cause the DTRACE table to be displayed.

Key-in Sequence	Comments
DTRACE RETURN	Displays all trace keywords and their contents on the screen, or the DTRACE table.
DTR RETURN	Displays all trace keywords and their contents.

The following example demonstrates how to enable a tracepoint display.

Key-in Sequence	Comments
TR0 = 48H ON RETURN	Tracepoint display information can be controlled by actions occurithin the user's prm. This command enables the tracepoint display if the contents at address 48H is executed.

The following screen display illustrates how the DTR and TRn commands affect the DTRACE table.


*DTR
TRACE DISPLAY CONTROLS (D TRACE)
TD= ON instruction display, enter ON or OFF
TR= OFF register display, enter ON or OFF
TBD= OFF TB1=OFF TB2=OFF TB3=OFF
TS= OFF status display, enter ON or OFF
DISPLAY START/STOP CONTROLS
TRD=OFF TR1=OFF TR2=OFF TR3=OFF
TV= OFF (tv=n value switch) (TRx=address sw)
*TR0 48H ON
*DTR
TRACE DISPLAY CONTROLS (DTRACE)
TD= ON instruction display, enter ON or OFF
TR= OFF register display, enter ON or OFF
TBD= OFF TB1=OFF TB2=OFF TB3=OFF
TS= OFF status display, enter ON or OFF
DISPLAY START/STOP CONTROLS
TRD= 0048H ON TR1=OFF TR2=OFF TR3=OFF
TV= OFF (TV=n value switch) (TRx=address sw)
*

Special Function Key 3

Holding the function key and pressing the number 3 also displays the DTRACE table, as shown in the following example.

Key-in Sequence	Comments
FUNCT 3	Displays all trace keywords and their contents.

Program Emulation Commands

The emulator software contains two commands that start emulation of the user's program, the STEP and the GO commands. Both commands check for breakpoints being set, but the STEP command performs the checking in software while the GO command uses hardware breakpoint checking. These commands are described in the following sections.

The STEP Command

The STEP command emulates the user's program one instruction at a time, checking for a trace enable command (TR*n* or TV) or a breakpoint (if they are enabled) after executing each instruction. The EMV-51A software emulates either a single instruction or a group of instructions. The emulation begins with the instruction pointed to by the program counter (PC), or at the address specified in the STEP command.

During STEP emulation, each instruction is executed and then checked for a trace enable command being set. If software breakpoint checking is enabled, the emulator software also checks for breakpoints being set. If a trace enable command is set, the emulator software displays up to four lines of trace information depending on the trace control commands that

are enabled. If a breakpoint is found and breakpoints are enabled, the emulator software stops the emulation and displays the address where the breakpoint occurred. Following are several command entry examples.

Key-in Sequence	Comments
STEP RETURN	Emulates the user program instruction pointed at by the program counter. Breakpoints are not enabled.
STEP FROM 50H RETURN	Emulates the user program instruction located at address 50H. Breakpoints are not enabled.
STEP 50H COUNT = 20H RETURN	Emulates user program instructions beginning at address 50H and continuing for the next 20H instructions inclusive. Breakpoints are not enabled.
STEP FROM 50H BR RETURN	Emulates user program instructions beginning at address 50H. The BR parameter enables software breakpoint checking.

The GO Command

The GO command emulates the user's program in real time beginning with the instruction pointed to by the program counter. If an address is entered with the GO command, the emulator software updates the program counter prior to beginning the emulation. Emulation continues until an address, branch, or range break occurs, or the emulation session is aborted by pressing the ESC key.

If address, branch, or range breakpoints are set prior to entering the GO command, the emulator hardware checks for breakpoints while executing each instruction. If a breakpoint is found, the emulation stops after the instruction executes and the EMV-51A displays thest add the rstopng tulation. Several GO command entry examples are as follows.

Key-in Sequence	Comments
GO RETURN	Begins emulation from the address contained in the program counter. If address, branch, or range breakpoints are set, the emulation stops if a breakpoint is encountered.
GO FROM 50H RETURN	Begins emulation from address 50H after updating the program counter. If address, branch, or range breakpoints are set, the emulation stops if a breakpoint is encountered.

NOTE

The emulator software issues one of the following warning messages if the corresponding conditions occur:

Warning DO — an interrupt occurs just before the breakpoint to halt emulation.

Warning D1 — a breakpoint occurs on the first instruction of an interrupt service routine.

Warning D2 — when interrupts are enabled, and an 8050 timer is enabled with an initial value of 0 or 1. A false timer overflow occurs and an interrupt is generated.

The P Command

The P command displays the instructions emulated by either the STEP or GO commands, entered as follows:

Key-in Sequence	Comments
P RETURN	Displays the last two instructions emulated.

If the command is issued before any instructions are emulated, the emulator software displays the message NO DATA.

Emulation Control Screen Displays

The demonstration program (see Figure 2-6) is used to demonstrate the emulation commands. The following examples show how to use the emulation and display commands. The following screen display illustrates the results as they occur on the screen.

Key-in Sequence	Comments
BRB = ON RETURN	Enables the branch on breakpoint.
GO FROM .DEMO RETURN	Emulates the user's program from the current address in the program counter until a breakpoint is reached. Then the emulator software displays the cause of the breakpoint and the address where it occurred.
P RETURN	Displays the last two instructions emulated.
BR RETURN	Displays the breakpoint command table.

```
*BRB=ON
*GO FROM .DEMO
.DEMO                  | 0040H=JBC   6CH,0002H      BRB
*P

.DEMO                  | 0202H=SUBB  A,R3           DASM
                       | 0040H=JBC   6CH,0002H      DASM
*BR
|-------------------------------------------------------------
              BREAKPOINT SETTINGS                  |     TYPE
|-------------------------------------------------|-----------
| BR0=ON     BR1=OFF     BR2=OFF     BR3=OFF      | location
| BRR =  OFF                                      | range
| BRB =  OFF (go mode only)    BC disables all    | branch
|   BV = OFF (step mode only)   breakpoints       | value
|-------------------------------------------------------------
```

Key-in Sequence	Comments
BR0=61 RETURN	Changes the contents of BR0 to 61H.
BR RETURN	Displays the updated contents of the breakpoint command table. These commands and the resulting screen displays are illustrated as follows.

```
*BR0=61

*

-----
*BR
|-------------------------------------------------------------------
|                BREAKPOINT SETTINGS                |      TYPE
|-------------------------------------------------|----------------
| BR0=0061H  BR1=0FF     BR2=0FF     BR3=0FF      | location
| BRR =  0FF                                      | range
| BRB =  0FF (go mode only)     BC disables all   | branch
|   BV = 0FF (step mode only)   breakpoints       | value
|-------------------------------------------------------------------
*

-----
```

The following examples show how to use the trace and STEP commands, and the following screen display illustrates the results as they occur on the screen.

Key-in Sequence	Comments
TD=ON RETURN	Enables the instruction trace.
TR=ON RETURN	Enables the register trace.
STEP FROM 61 BR RETURN	Enters the STEP command with breakpoints enabled. The EMV-51A software controls the hardware emulation of the user's program and stops the emulation when a breakpoint is encountered. Instruction trace (TD) and register trace (TR) information are displayed (see the following screen display).

```
*TD=ON
*TR=ON
*STEP FROM 61 BR
                              | 0061H=MOV   13H,R7          STEP
ACC=00H PSW=00H R0=00H R1=00H R2=50H R3=B3H R4=50H R5=01H
R6=50H R7=01H
```

CHAPTER 8
USING ADVANCED COMMANDS

Introduction

This chapter describes the macro and conditional command groups. The following sections describe the commands that comprise each group, discuss command use, and give command entry examples. The following topics are discussed:

- Using macros
- Using conditional commands
- Macro and conditional commands demonstration

Using Macros

A macro is a set of frequently used commands. When defined as a macro, the command string is given a name. The command string can then be called into use by the single macro name. The macro can have unique parameter values passed to it each time it is executed.

This section describes the commands that define a macro, display a macro, save a macro definition, execute a defined macro, and remove a macro.

Defining Macros

The following sections describe how to define a macro, how to terminate the definition, how to pass parameters, and how to include messages that will appear on the screen to indicate that the commands in the macro have been executed properly.

The DEFINE Command

The DEFINE command assigns a single name (the macro name) to a group of commands (the macro definition). The name is up to 32 characters long with a colon (:) as the first character. After entering the DEFINE command, the emulator software response is a prompt consisting of a period immediately followed by an asterisk (.*).

All user entry, following the prompt, is placed in the new macro definition.

NOTE

You cannot nest the DEFINE command in a DEFINE command. The following example will cause an error when executed:

```
DEFINE :WORK
.*DEFINE :SUBROUTINE
.*DBYTE %0 TO %1
.*CBYTE 0 TO 10T
.*EM
```

The EM Command

The EM command terminates a macro definition. Entering the EM command causes the emulator software to close the macro definition and once again display the single asterisk prompt.

Parameter Passing

Customizing a macro definition is accomplished by passing up to 10 parameters to the commands making up the definition. To indicate the points that receive parameter strings, place a parameter indicator (a percent sign (%) followed by a number from 0 to 9) at the desired points. The actual parameter strings are passed to the definition via the macro execution command, described later in this chapter. The following example illustrates setting up the definition to receive parameter values.

Key-in Sequence	Comments
DEFINE :SHOW RETURN	Defines a new macro definition called SHOW.
DBYTE %0 TO %1 RETURN	Displays the contents of a range of addresses within data memory. The actual range addresses are passed with the macro execution command.
EM RETURN	Terminates the macro definition.

The following screen display illustrates how the DEFINE and EM commands will appear on the screen. A macro is defined, and receives parameter values when executed.

```
*DEFINE :SHOW
. *DBYTE %0 TO %1
. *EM
*
```

NOTE

Commands may be entered to a macro in upper or lower case. However, the system internally converts all the entered characters to upper case.

Writing Messages

The WRITE command can be inserted in a macro definition to display messages on the screen. With this command, the result of an expression or a string of characters is displayed. The following example illustrates how to use the WRITE command.

Key-in Sequence	Comments
WRITE 'THE RESULT IS' RETURN	Displays the message THE RESULT IS when the command is executed.

WRITE .ABLE + 100H RETURN	Displays the result of the expression .ABLE + 100H when the command is executed.
WRITE 'THIS IS RETURN	When a message does not end with an apostrophe, a double asterisk prompt will appear until an ending asterisk completes the message.
A THREE LINE RETURN	This will be the second line in the message.
MESSAGE' RETURN	This will be the third line in the message.
WRITE 'I''M DONE' RETURN	An embedded apostrophe will appear as one apostrophe when the message is displayed.

The following screen display demonstrates the usage of the WRITE command.

```
*WRITE 'THIS IS
**A THREE LINE
**MESSAGE
THIS IS
A THREE LINE
MESSAGE
*
*WRITE 'I''M DONE
I'M DONE
*
```

Displaying Macro Information

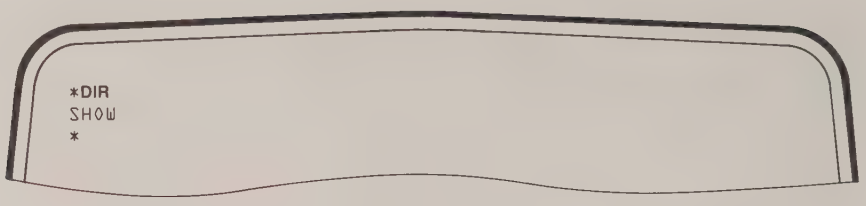
The EMV-51A software contains two commands, DIR and MACRO, that display the names of macros currently in memory, or the text of specified macro definitions. These commands are described in the following sections.

The DIR Command

The DIR command displays a directory of defined macro names currently residing in memory. A macro definition must be user defined or loaded from a disk file before the DIR command can display its name. The directory command is entered as follows:

Key-in Sequence	Comments
DIR RETURN	Displays a directory of all defined macros.

If :SHOW has been defined, entering the DIR command will result in the following screen display.

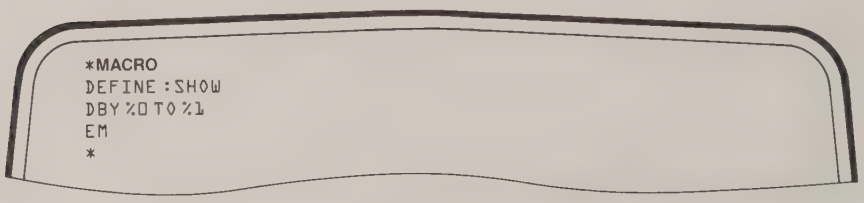


The MACRO Command

The MACRO command displays the definitions of one, several, or all defined macros currently residing in memory. A macro definition must be user defined or loaded from a disk file before the MACRO command can display its definition. The MACRO command is entered as follows:

Key-in Sequence	Comments
MACRO :NULL RETURN	Displays the text definition of the macro :NULL.
MACRO :NULL,:DEMO RETURN	Displays the text definitions of the two macros :NULL and :DEMO.
MACRO RETURN	Displays the text definitions of all defined macros.

Entering the MACRO command will result in the following screen display.



Saving Macros (The PUT Command)

The PUT command saves a copy of the macro definition, including the DEFINE and EM commands, in a disk file. The macro definition can be retrieved at a later time using the INCLUDE command, described later in this chapter. The following examples demonstrate entering the PUT command.

Key-in Sequence	Comments
PUT :F1:MACRO.SAV :NULL RETURN	Saves a copy of the macro :NULL in the file labeled MACRO.SAV on disk drive :F1:.
PUT :F3:MACRO.SAV :DEM,:RUN RETURN	Saves a copy of the macros :DEM and :RUN in the file labeled MACRO.SAV on disk drive :F3:.

PUT :F3:SAVE.IT MACRO RETURN

Saves a copy of all macros currently in memory in the file labeled SAVE.IT on disk drive :F3:.

NOTE

The PUT command destroys the contents previously in the disk file. If one file is used for all macro definitions, retrieve all macro definitions from the file prior to using the PUT command.

Executing Macros

The following sections show how to execute a macro and how to list a macro definition.

The Macro Execution Command

A macro is executed by entering the macro name preceded by a colon (:). If the macro requires parameters for execution, the parameters are entered with the macro name, separated by commas or spaces. The parameter values then replace the parameter indicators (%0 through %9) sequentially.

The following examples demonstrate entering a macro name with and without parameter values.

Key-in Sequence	Comments
:DIGIT 3,44 RETURN	Executes the macro DIGIT and supplies the parameters 3 and 44.
:MULT .MUL1,.MUL2 RETURN	Executes the macro MULT and supplies the parameters represented by the symbols .MUL1 and .MUL2.
:NULL RETURN	Executes a macro called NULL without parameters.
:DEMO RETURN	Executes a macro called DEMO without parameters.
:SHOW 0,10 RETURN	Executes a macro called SHOW with the parameters 0 and 10.

Entering the macro :SHOW with the required parameters will result in the following screen display.

```
*:SHOW 0,10H
.ROX
0000H=04H 75H FEH 3AH A7H EAH 6EH 86H 00H 01H 02H 94H 04H
8EH CAH ECH
0010H=6EH
```

The ENABLE/DISABLE Macro Expansion Commands

When a macro definition is executed, the emulator software may or may not list the definition prior to executing it. The default condition is that macro definitions are not listed prior to execution of the macro.

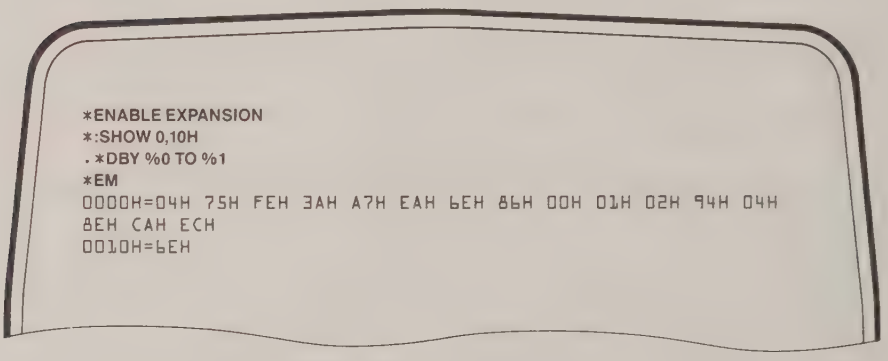
To enable the display of macro definitions before executing the macro, enter the following command:

Key-in Sequence	Comments
ENABLE EXPANSION RETURN	Enables the display of the macro definition prior to the execution of the macro.

Entering ENABLE EXPANSION and a macro name will result in the following screen display.

NOTE

In most cases, sample memory display values will differ from your displayed values.



To prevent the display of macro definitions, enter the following command:

Key-in Sequence	Comments
DISABLE EXPANSION RETURN	Stops the display of the macro definition prior to the execution of the macro.

Deleting Macros (The REMOVE Command)

The REMOVE command deletes macro names from memory, as shown in the following examples:

Key-in Sequence	Comments
REMOVE :NULL RETURN	Deletes the macro NULL from memory.

REMOVE :NULL.:DEMO Deletes the macros NULL and DEMO from memory.

RETURN

REMOVE MACRO Deletes all macro names from memory.

RETURN

Command Files

A command file consists of valid EMV-51A commands created with either the ISIS-iPDS editor or the DEFINE macro command and stored in a disk file. For additional information on creating command files, refer to the *iPDS™ User's Guide*, order number 162606. The following sections describe how to use the INCLUDE command to execute command files and macro command files.

Executing Command Files

The INCLUDE command gets a command file from the specified disk file and executes the contents sequentially as they appear in the file. Following is an example of entering the INCLUDE command.

Key-in Sequence	Comments
INCLUDE :F1:TEST.V1 RETURN	Loads a command file called TEST.V1 from device :F1:.

Executing Macro Command Files

Macro definition commands, previously saved with the PUT command, are also loaded with the INCLUDE command and executed. Since the first command in the macro is the DEFINE command, the macro is re-DEFINED in memory and available for future reference. Following is a command entry example.

Key-in Sequence	Comments
INCLUDE :F1:MACRO.SAV RETURN	Loads a command file called MACRO.SAV from device :F1:.

As the macro is loaded into program memory, the emulator software automatically lists the macro definition. The DISABLE EXPANSION command has no effect during execution of the INCLUDE command. Entering the INCLUDE command with a command file will result in the following screen display.

```
-INCLUDE :F0:MACRO.SAV
-DEFINE SHOW
. -DBY %0 TO %1
. -EM
-
```

Using Compound Commands

Compound commands provide conditional testing and looping capabilities. Conditional testing uses the IF..THEN..ELSE command, while the looping controls use both the REPEAT..WHILE/UNTIL and COUNT..STEP..WHILE/UNTIL commands. The following sections cover:

- Setting up the conditional IF command
- Setting up the REPEAT command
- Setting up the COUNT command

Setting Up the Conditional IF Commands

The IF..THEN..ORIF..ELSE command sequence permits conditional testing and execution of groups of EMV-51A commands. The IF command can be nested to any depth. The following command entry examples show several methods of using the IF command.

In the first command entry example, the command STEP is executed if the contents of symbolic name .LOOP is equal to 5. If the test is false, the command GO is executed. Chapter 10 contains additional information about the IF command.

```
IF .LOOP = 5 THEN        If .LOOP = 5 then execute
   STEP                  the STEP command.
ELSE                     Otherwise, execute
   GO                    the GO command.
ENDIF                    Terminate the IF..LOOP.
```

In the second command entry example, the value of the symbolic label .T determines the sequence of a series of tests. If the initial value of .T is less than 3, the testing begins. If the value of .T is greater than 1, phase B is performed. If the value of .T equals 1, phase C is performed.

```
IF .T < 3 THEN
   .ABLE = 10*.T
   .T = .ABLE/10
   WRITE 'PHASE A PASSED'
   IF .T > 3 THEN
        .BAKER = .ABLE*.T/4H
        .T = .BAKER/.ABLE
        WRITE 'PHASE B PASSED'
        STEP FROM .T COUNT = .ABLE
        IF .T = 1 THEN
             STEP FROM .T COUNT = .BAKER
             WRITE 'ALL PHASES PASSED'
      ELSE
             WRITE 'PHASE C FAILED'
      END
      END
   ELSE
      WRITE 'TEST FAILED'
      STEP FROM .T BR
   ENDIF
```

The following screen display demonstrates a typical application of the conditional IF command. The command is used to emulate various sections of a program depending on the contents of a specified register.

```
*DEFINE :WORK2
. *DBY 3E = 40
. *DEFINE .Z = 1
. *REP
. *WHILE .Z > 0
. *STEP FROM (DBY 3E)
. *IF DBY 3E = 48 THEN
. *   WRITE 'BYTE TAKEN'
. *   ORIF (DBY 3E) = 54
. *   WRITE 'ASCII STORED'
. *   ORIF (DBY 3E) = 61
. *   .Z = .Z-1
. *   DBY 6F TO 74
. *   ENDIF
. *DBY 3E = (DBY 3E) + 1
. *END
. *EM
```

The macro :WORK demonstrates how the REPEAT, WHILE, STEP, IF...ORIF, and WRITE commands can be combined to emulate the portions of a program. Messages are displayed when various points in the program are emulated.

Setting Up the REPEAT Command

The REPEAT..WHILE and REPEAT..UNTIL command sequences execute a series of commands conditionally. The REPEAT..WHILE sequence executes a series of commands while a specified condition is true. The REPEAT..UNTIL sequence executes a series of commands until a specified condition becomes true. The UNTIL and WHILE clauses can appear once, never, or any number of times, anywhere in the loop.amples illustrate both forms of the REPEAT command.

```
REPEAT
        WHILE DBYTE 0 > 55H
        STEP FROM 10H COUNT = 10H
        DBYTE 0 = (DBYTE 0) - 1
END

REPEAT
        UNTIL DBYTE 0 = 55H
        STEP FROM 10H COUNT = 10H
        DBYTE 0 = (DBYTE 0) + 1
END
```

The first REPEAT begins a loop that repeats while memory location 0 in data memory is greater than 55H. The loop executes STEP on each pass, then decrements the memory location.

The second REPEAT begins a loop that repeats until memory location 0 in data memory equals 55H. The loop executes STEP on each pass, then increments the memory location.

The following screen display demonstrates a typical application of the REPEAT command. The command emulates a program until a memory location equals a specified value.

```
*DEFINE :WORK
. *DBY 3E=40            ;DBYTE 3E=40
. *DBY 3F=0
. *REP
. *UNTIL (DBY 3F)=3
. *GO FROM .DEMO
. *DBY 3F
. *DBY 6F TO 75
. *DBY 3F=(DBY 3F)+1
. *END
. *EM
```

Macros can reference data memory for function arguments. In the previous display, the UNTIL and DBY commands get their arguments from data memory. The following screen display shows the execution of this macro.

```
*BRO=.DONE
*:WORK
.DONE           006BH=POP       .ROX       GO-BREAK
ACC=3FH PSW=80H R0=FFH R1=75H R2=00H R3=FFH R4=E3H R5=01H
R6=FFH R7=8CH
   CARRY=1      AUX=0      FLAG=0 RBS=00  OVERFLOW=0 UTL=0 PAR=0

DBYTE .ROX+003FH=00H
.BUFFER
006FH=38H  3FH  36H  3FH  37H  3FH  31H
DONE            006BH=POP       .ROX       GO-BREAK
ACC=3FH PSW=80H R0=FFH R1=75H R2=00H R3=FFH R4=E3H R5=01H
R6=FFH R7=8CH
   CARRY=1      AUX=0      FLAG=0 RBS=00  OVERFLOW=0 UTL=0 PAR=0

DBYTE .ROX+003FH=01H
.BUFFER
006FH=38H  3FH  36H  3FH  37H  3FH  31H
DONE            006BH=POP       .ROX       GO-BREAK
ACC=3FH PSW=80H R0=FFH R1=75H R2=00H R3=FFH R4=E3H R5=01H
R6=FFH R7=8CH
   CARRY=1      AUX=0      FLAG=0 RBS=00  OVERFLOW=0 UTL=0 PAR=0

DBYTE .ROX+003FH=02H
.BUFFER
006FH=38H  3FH  36H  3FH  37H  3FH  31H
```

Setting Up the COUNT Command

The COUNT..WHILE and COUNT..UNTIL command sequences execute a series of commands a specified number of times. The WHILE and UNTIL clauses permit conditional termination of the COUNT command sequence depending on conditions within the program.

The following example illustrates the WHILE form of the COUNT command sequence.

```
COUNT 12H
        WHILE DBYTE 0>55H
        STEP FROM 10H COUNT=10H
END
```

COUNT 12H begins a loop that repeats 12H times and while memory location 0 in data memory exceeds 55H. The loop executes STEP on each pass, with the code presumably altering the contents of data memory location 0. END terminates the COUNT command.

The following example illustrates the UNTIL form of the COUNT command.

```
COUNT 12H
        UNTIL DBYTE 0=>355H
        STEP FROM 10H COUNT=10H
```

COUNT 12H begins a loop that repeats 12H times or until data memory location 0 equals or exceeds 55H. The loop executes STEP on each pass, with the code presumably altering the contents of data memory location 0. Again, END terminates the COUNT command.

Using Special Function Keys

The emulator software recognizes 10 special function keys numbered 0 through 9. These keys are entered by typing the corresponding number while holding down the FUNCT key similar to the way a SHIFTed character is typed. Function keys 1 through 4 are predefined and are described in Chapters 4 through 6. Function keys 0 and 5 through 9 are user-defined. These keys can be assigned to a macro name if the macro does not require parameters passed to it. These predefined and user defined function keys can also be used by pressing only the numbered key followed by a carriage return.

Assigning Function Keys to Macros

The following examples demonstrate the procedure to assign a function key to a macro definition. Macro names associated with functions keys can be from 1 to 9 characters in length.

FUNC=0 :DEMO

Assigns function key 0 to the macro definition labeled :DEMO.

FUNC=5 :NULL

Assigns function key 5 to the macro definition labeled :NULL.

Function Key Precautions

Macro definitions can be assigned to function keys 0 through 9, but 0 and 5 through 9 should be assigned first. If function keys 1 through 4 are re-defined, retrieval of the system-defined function is not easy. If this does occur, however, the system definition for function keys 1 through 4 can be retrieved by one of the following methods:

1. EXIT the emulation session and re-load the EMV-51A software by entering EMV51A. This procedure retrieves the original function key definitions, but destroys all programs currently loaded.

2. Define a macro that duplicates the original function and assign the desired function key to the macro definition.

For example, if function key 4 which generates the register display has been re-defined, use the following procedure to retrieve the original definition.

1. Define a macro that duplicates the original function:

DEFINE :REG RETURN — Defines a macro labeled :REG that duplicates the original REG function key.

EM RETURN — Displays the register keywords and their contents when executed.

2. Assign the desired function key to the new macro.

FUNC = 4 :REG RETURN — Assigns function key 4 to the macro :REG that re-creates the original function.

Displaying Assigned Function Keys

To display all assigned function keys enter the following command:

FUNCTION RETURN — Displays all assigned function keys.

Advanced Commands Demonstration

This section demonstrates a practical use of the macro and conditional commands by defining a macro that uses conditional commands to emulate the demonstration program generated in Chapter 2.

```
*DEFINE :MESS
.*WRITE'
.**LOAD NEXT PROGRAM
.**'
.*EM
*MESS

LOAD NEXT PROGRAM
*
```

In this screen display, the macro :MESS is defined as follows:

Key-in Sequence	Comments
DEFINE :MESS [RETURN]	DEFINE initiates the definition of the macro :MESS.
WRITE ' [RETURN]	The WRITE command can be used to display messages on the screen. The initial quotation mark indicates the following characters will be displayed.
LOAD NEXT PROGRAM	This is the message that will be displayed.
[RETURN]	
['] [RETURN]	This quotation mark indicates the end of the message to be displayed.
EM [RETURN]	This terminates the macro definition.
:MESS [RETURN]	Displays the macro definition.

The following screen display shows how macro definitions can include other macro definitions. In this example, the macro :TEST consists of the macros :WORK and :MESS.

```
*DEFINE :TEST
. *:WORK
. *:MESS
. *EM
*
```

The following screen display shows how to assign the macro :TEST to function key 0.

```
*FUNC=0 :TEST
*#0
*
```

The actual key-in sequence is as follows:

FUNC=0 :TEST [RETURN]

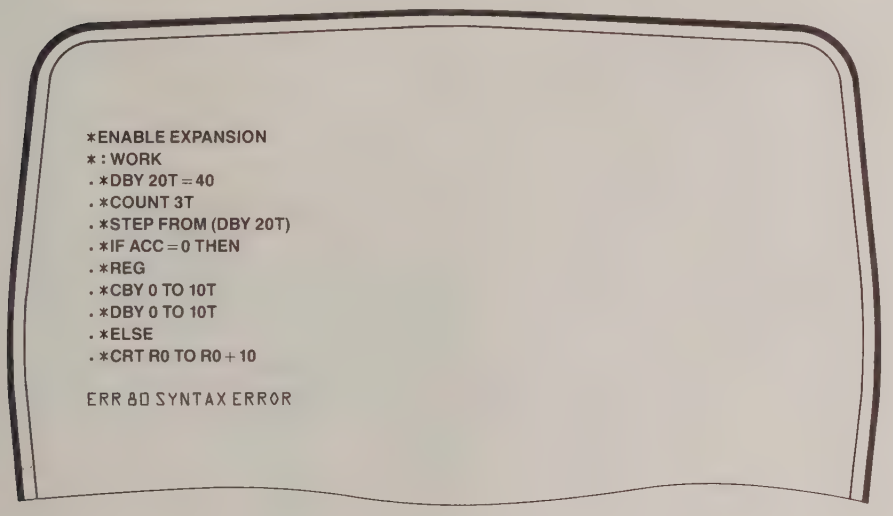
Once the 0 function key has been assigned, it can be accessed by pressing the FUNCT key and the number 0 at the same time. When this is done, the following screen display will be produced.

```
.DONE                    006BH=P0P        .R0X                  G0-BREAK
ACC=3FH PSW=80H R0=FFH R1=75H R2=00H R3=FFH R4=E3H R5=01H R6=FFH R7=8CH
   CARRY=1     AUX=0     FLAG=0 RBS=00     OVERFLOW=0 UTL=0 PAR=0

DBYTE .R0X+003FH=00H
.BUFFER
006FH=38H 3FH 36H 3FH 37H 3FH 31H
DONE                     006BH=P0P        .R0X                  G0-BREAK
ACC=3FH PSW=80H R0=FFH R1=75H R2=00H R3=FFH R4=E3H R5=01H R6=FFH R7=8CH
   CARRY=1     AUX=0     FLAG=0 RBS=00     OVERFLOW=0 UTL=0 PAR=0

DBYTE .R0X+003FH=01H
.BUFFER
006FH=38H 3FH 36H 3FH 37H 3FH 31H
DONE                     006BH=P0P        .R0X                  G0-BREAK
ACC=3FH PSW=80H R0=FFH R1=75H R2=00H R3=FFH R4=E3H R5=01H R6=FFH R7=8CH
   CARRY=1     AUX=0     FLAG=0 RBS=00     OVERFLOW=0 UTL=0 PAR=0
DBYTE .R0X+003FH=02H
.BUFFER
006FH=38H 3FH 36H 3FH 37H 3FH 31H

LOAD NEXT PROGRAM
*
```

Before the macro :WORK can be redefined, it must be removed. In the following screen display, the previously defined macro :WORK is removed and a new macro :WORK is defined. Notice that there is an error following ELSE.

```
*REMOVE :WORK
*DEFINE :WORK
. *DBY 20T = 40
. *COUNT 3T
. *STEP FROM (DBY 20T)
. *IF ACC = 0 THEN
. *REG
. *CBY 0 TO 10T
. *DBY 0 TO 10T
. *ELSE
. *CRY R0 TO R0 + 10
. *DBY R0 TO R0 + 10
. *ENDIF
. *DBY 20T = (DBYT) + 1
. *END
. *EM
```

If you tried to execute this macro, ERR 80 SYNTAX ERROR would be displayed. To identify the error quickly, enable the expansion feature and execute :WORK again, as shown in the following screen display.



When macro expansion is enabled, the emulator software displays a macro definition prior to executing the macro. The last line displayed contains the error that prevented the macro from executing. To correct this error, remove the macro and define it correctly. For longer macros, use the PUT command to save the macro definition on disk and then use a text editor to modify it.

CHAPTER 9
EMV-51A REFERENCE MATERIAL

Introduction

This chapter contains reference material on the EMV-51A emulator and the MCS-51 microcontroller family. It lists the keywords used by the emulator software and defines the MCS-51 operators used in the EMV-51A software.

EMV-51A Symbols

The emulator software contains a table of predefined system symbolic names, known as system symbols. Table 9-1 lists the symbols and gives a definition of each.

Table 9-1 System Symbols

Keyword	Definition
.ACC	Accumulator
.B	Multiplication register address
.P0	Port 0
.SP	Stack pointer
.DPL	Data pointer, low byte
.DPH	Data pointer, high byte
.TCON	Timer/counter port 0
.TMOD	Timer mode
.TL0	Timer 0 low byte
.TL1	Timer 1 low byte
.TH0	Timer 0 high byte
.TH1	Timer 1 high byte
.P1	Port 1
.SCON	Serial port control
.SBUF	Serial port buffer
.P2	Port 2
.IE	Interrupt enable
.P3	Port 3
.IP	Interrupt priority
.PSW	Program status word

In addition to the system symbols, the EMV-51A software uses several one-character keywords as punctuation marks. These are listed in Table 9-2.

Table 9-2 Punctuation and Delimiter Keywords

Keyword	Meaning
.	Identifies symbolic reference
,	Separates items in a list
( )	Controls order of evaluation in expressions
+	Indicates a positive value or addition
–	Indicates a negative value or subtraction
*	Serves as the EMV-51A prompt character
=	Assigns one value to another
>	The greater than operator sign
<	The less than operator sign
"	Encloses opcode mnemonic constants in expressions
'	Encloses string characters
:	Identifies a macro name
%	Identifies formal parameters in macro definitions

EMV-51A Operators

The EMV-51A software supports four types of operators: arithmetic, content, relational, and logical operators. An operator is a category of commands that perform specific functions. As an example, the content operator contains the commands that access the various memory regions in the 8051 memory map. The following sections describe the four categories and list the operators in each.

Arithmetic Operators

Arithmetic operators consist of keywords that specify the five types of arithmetic operations recognized by the emulator software. Each arithmetic operator returns a 16-bit value. If the operation involves an 8-bit register as the destination the register receives the low-order eight bits and the upper eight bits are truncated. Table 9-3 lists the arithmetic operators.

Table 9-3 Arithmetic Operators

Operator	Operation
+	Unary plus
–	Unary minus (2's complement)
*	Multiplication
/	Integer division
MOD	Modulo division
+	Addition
–	Subtraction

Content Operators

Content operators consist of keywords that reference specific types of memory addressed by the microcontroller. Table 9-4 lists the content operator keywords, while the following sections describe each memory space.

Table 9-4 Content Operator Operators

Operator	Type of Memory
CBYTE	Code memory
DBYTE	On-chip data memory
RBYTE	Special function register memory
PBYTE	External data memory
RBIT	Bit-addressable memory

Code Memory (CBYTE)

The MCS-51 microcontroller family can address 64K bytes of program memory. The 8051 and 8751 microcontrollers separate program memory into 4K bytes of internal memory and 60K bytes of external program memory, while the 8031 provides 64K bytes of external program memory.

The emulator hardware contains 8K bytes of RAM that can optionally replace the 4K bytes of internal program memory and 4K bytes of external program memory of the 8051/8751 microcontrollers. This replacement is controlled by moving the jumper in the controller as described in Chapter 2. The 8K bytes of RAM can replace the internal 4K bytes of memory space, the 4K bytes of internal plus the 4K bytes of external memory, or none of the memory space of the 8051/8751 microcontrollers. Setting the jumper in the controller box as described in Chapter 2 increases the code memory capacity to 8K bytes of RAM.

CBYTE Restrictions

When using the emulator hardware 4K RAM as the microcontroller ROM, a program containing up to 64K bytes of instructions can be emulated and debugged. In the external 60K bytes of user program memory, breakpoints can be set and programs can be emulated, but program code cannot be modified.

If the jumper in the controller box remains as shipped, a program longer than 4K bytes is emulated and debugged in blocks of up to 4K instructions. The memory addresses for each block must be modified to load into the EMV-51A's 4K RAM. As each block is emulated and debugged, it can be programmed into a PROM, and inserted into the external 60K of program memory space. In this manner, the entire user program can be emulated and debugged. However, if the user enables external memory from 0 to 4K, the CBYTE command is limited to reading memory contents. It cannot modify program instructions in external memory.

If the jumper in the controller box is configured for 8K bytes of code memory, a program longer than 8K bytes is emulated and debugged in blocks of up to 8K instructions. In this case, the memory addresses for each block must be modified to load into the configured 8K RAM. Again, as each block is emulated and debugged, it can be programmed into a PROM and inserted into the external 60K of program memory space, and the entire user program can be emulated and debugged.

On-Chip Data and Register Memory (DBYTE, RBIT, and RBYTE)

The MCS-51 microcontrollers contain 128 bytes of internal data memory organized as 32 general-purpose registers, 16 bytes of bit-addressable memory, and 80 bytes of general-purpose RAM. In addition, the microcontrollers contain a block of memory that consists of input/output registers and various internal control registers. Figure 9-1 illustrates the organization of the internal memory.

On-Chip Data Memory (DBYTE and RBIT). The entire 128 bytes of internal data memory can be accessed with the DBYTE keyword. The RBIT keyword is used to access the 16 bytes of bit-addressable memory on a bit-by-bit level and bit-addressable portions of the 8050 registers as shown in Figure 9-1 and Table 9-5.

Special Function Register Memory (RBYTE). The MCS-51 microcontroller family contains a block of memory that consists of the input/output registers and various internal control registers. The emulator software assigns system symbolic names to the registers in this memory block. The pre-defined registers can be accessed by entering the RBYTE keyword followed by an address or system symbol. Table 9-1 lists the contents, system symbol, and a brief description of the register memory locations. Refer to the *MCS®-51 Macro Assembler User's Guide*, order number 9800937, for additional information.

External Data Memory (PBYTE)

The MCS-51 microcontroller family isolates program instructions from program data by providing separate 64K-byte blocks of program memory and data memory. The PBYTE keyword accesses the 64K bytes of external data memory.

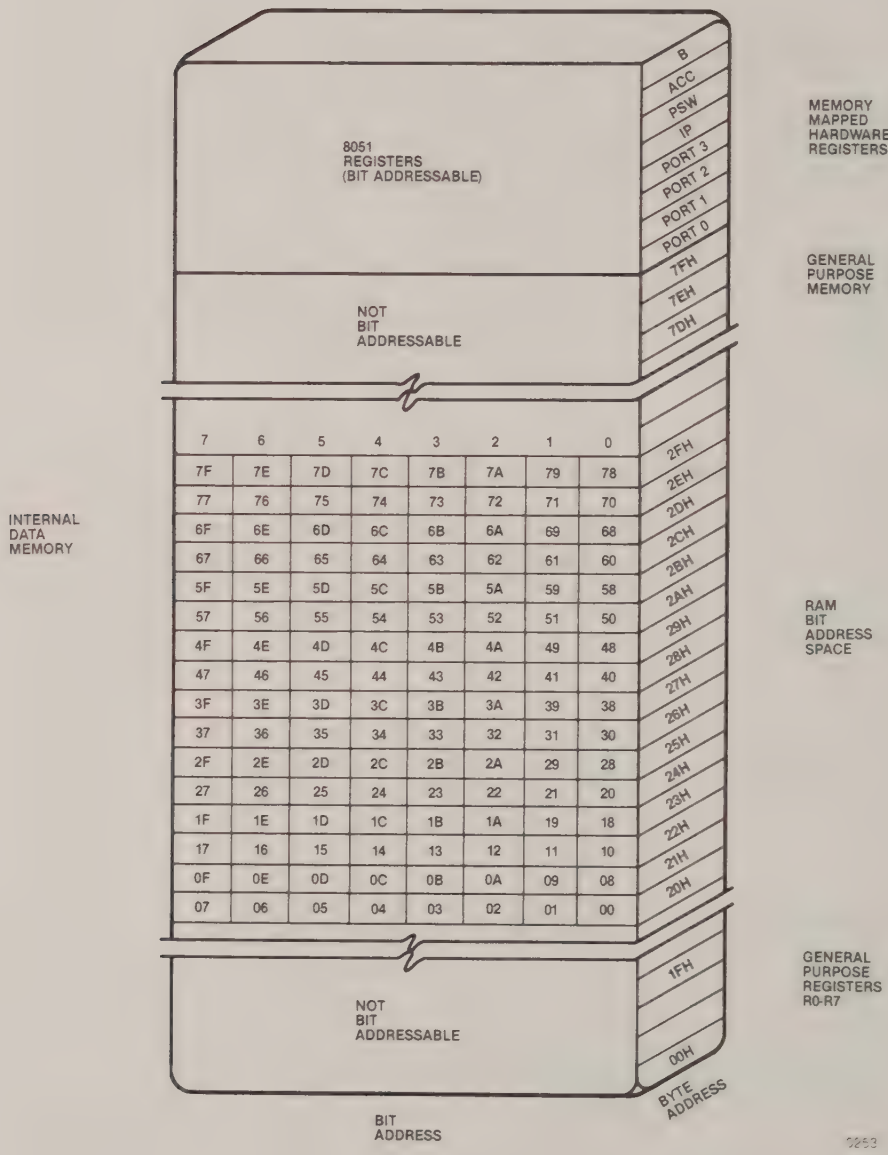


Figure 9-1 Internal Data Memory Organization

Table 9-5 Bit-Addressable Register Memory Contents

Hex Address	System Symbol	Meaning
80H-87H	.P0	Port 0
88H	.IT0	Timer control
89H	.IE0	Timer 0 interrupt edge flag
8AH	.IT1	Timer 1 interrupt type control bit
8BH	.IE1	Timer 1 interrupt edge flag
8CH	.TR0	Timer 0 run control bit
8DH	.TF0	Timer 0 overflow flag
8EH	.TR1	Timer 1 run control bit
8FH	.TF1	Timer 1 overflow flag
90H-97H	.P1	Port 1
98H	.RI	Receive interrupt flag
99H	.TI	Transmit interrupt flag
9AH	.RB8	Receive bit 8
9BH	.TB8	Transmit bit 8
9CH	.REN	Receiver enable
9DH	.SM2	Serial mode control bit 2
9EH	.SM1	Serial mode control bit 1
9FH	.SM0	Serial mode control bit 0
A0H-A7H	.P2	Port 2
A8H	.EX0	Enable external interrupt 0
A9H	.ET0	Enable timer 0 interrupt
AAH	.EX1	Enable external interrupt 1
ABH	.ET1	Enable timer 1 interrupt
ACH	.ES	Enable serial port interrupt
B0H	.RXD	Serial port receive pin
B1H	.TXD	Serial port transmit pin
B2H	.INT0	Interrupt 0 input pin
B3H	.INT1	Interrupt 1 input pin
B4H	.T0	Timer/counter 0 external flag
B5H	.T1	Timer/counter 1 external flag
B6H	.WR	Write data (for external memory)
B7H	.RD	Read data (for external memory)
B8H	.PX0	Priority of external interrupt 0
B9H	.PT0	Priority of timer 0 interrupt
BAH	.PX1	Priority of external interrupt 1
BBH	.PT1	Priority of timer 1 interrupt
BCH	.PS	Priority of serial interrupt
D0H	.P	Parity flag
D2H	.OV	Overflow flag
D3H	.RS0	Register bank select bit 0
D4H	.RS1	Register bank select bit 1
D5H	.F0	Flag 0
D6H	.AC	Auxiliary carry flag
D7H	.CY	Carry flag
E0H-E7H	.ACC	Accumulator
F0H-F7H	.B	Multiplication register

Relational Operators

Relational operators consist of keywords that indicate the desired relational comparison to be performed. Table 9-6 lists the relational operators.

Table 9-6 Relational Operators

Operator	Relation
=	Is equal to
>	Is greater than
<	Is less than
≥	Is greater than or equal to
≤	Is less than or equal to
≠	Is not equal to

Logical Operators

Logical operators consist of keywords that specify a logical combining of numbers. Table 9-7 lists the logical operators.

Table 9-7 Logical Operators

Operator	Operation
NOT	1's complement
AND	Bitwise AND
OR	Bitwise OR
XOR	Bitwise exclusive OR

CHAPTER 10
COMMAND DICTIONARY

Introduction

Chapter 10 presents the information necessary to understand the EMV-51A commands. Entering and editing the commands is described, the form of the EMV-51A commands, command format notations, and definitions of terms used in the command format descriptions are given. This information is designed for the more experienced user of the EMV-51A emulation vehicle. The following topics are covered:

- Entering commands
- Form of EMV-51A commands
- Command description formats
- Special command format terms
- Alphabetical listing of commands

Entering Commands

The EMV-51A operations are controlled by command lines. A command line is a sequence of one or more command words separated by spaces and ending with a carriage return. The command line is normally limited to 122 characters. In most cases, these command words can be abbreviated to one, two, or three characters. The emulator software can receive command lines from four sources:

1. Commands entered from the keyboard.

2. Previously defined command files containing valid EMV-51A commands. After loading the EMV-51A software, the command file is loaded and executed using the INCLUDE command (described in Chapter 6).

3. Macros, defined as a sequence of related commands, stored in a disk file. After loading the EMV-51A software, the macro definition is loaded and executed using the INCLUDE command (macros and the INCLUDE command are described in Chapter 6).

4. Previously defined files loaded and executed using the SUBMIT command. Refer to the *iPDS™ User's Guide* for additional information on the SUBMIT command.

Specifying the Workfile

After entering the emulator software invocation command (EMV51A), the operating system automatically sets a workfile on the drive containing the EMV-51A software. This workfile is used for temporary storage of macro definitions. While entering the invocation command, an optional drive number for the workfile can be specified. The following command demonstrates specifying another drive as the location of the workfile:

:F1:EMV51A WORKFILE (:F*n*:)

In this example, the emulator software disk is inserted in drive :F1: and the workfile is specified as being on drive :F*n*:, where *n* is replaced by 0, 1, 2, 3.

Command Line Execution

The carriage return causes the command to be interpreted and, if no syntax errors are detected, executed immediately, with six exceptions. The exceptions are macro commands, continuation lines, the REPEAT command, the COUNT command, the WRITE command, and the IF command. Macro, REPEAT, COUNT, WRITE, and IF commands are described in Chapter 8. Continuation lines are described in the following section.

Continuation Lines

If a command line exceeds 122 characters, the line must be divided into multiple lines, known as continuation lines. Continuation lines are created by entering an ampersand character (&) and a carriage return prior to the 122nd character.

When entering the ampersand and carriage return, the emulator software responds by displaying a double asterisk prompt (**). The double prompt indicates the EMV-51A software is accepting a command line that exceeds the 122 character maximum length. The command line is not executed until termination of a continuation line with just a carriage return. At this point, the emulator returns to displaying a single asterisk prompt character.

NOTE

If the continuation character (&) appears in a comment line, it is interpreted as a part of the comment.

Entry Editing and Display Control

The emulator software recognizes control characters that permit editing of command line entries and controlling of the display generated by command execution. These control characters are described in the following sections.

Command Line Editing

The following control characters are used to correct errors in command line entry. These control characters consist of one or two keyboard characters. If a character is preceded by the word CTRL, press the CTRL key and simultaneously press the specified character. Otherwise, enter the indicated key.

Aborts any operation that is executing or aborts any command being entered.

Signifies the end of the command line.

Pressing the RUBOUT key deletes the last character typed. One character is deleted each time the key is pressed.

Gets the next keyboard entry and places the character in the command line buffer without interpreting it.

Echos the command line being entered.

Deletes the current line only, not any previous lines. If several continuation lines have been entered, the CTRL-X deletes only the current portion.

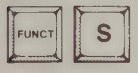
Deletes an entire command line, including all continuation lines.

Controlling the Display

Information is displayed on the screen one line at a time, scrolling the displayed line upward one line, and displaying another line. The following characters control the output to the display. Refer to the *iPDS™ User's Guide* for a complete explanation of control character functions.

Stops the output of new information to the screen.

Continues the output of new information to the screen.



Slows down or speeds up the screen display. Pressing FUNCT-S once slows the display down, while pressing it again speeds up the display.

Form of EMV-51A Commands

The command format requirements, listed later in this chapter, use special command format notations terms. This section lists the format notations and defines the command notation terms.

Notational Conventions

Because of the many different ways that a single command can be entered, it is not possible or desirable to list every correct entry. Instead, the general format of the command is described using special symbols or notational formats.

Command format notation consists of symbols adopted to help describe EMV-51A commands. These symbols are not part of the command but are used to precisely describe the format of the command.

The special characters used in these formats have no significance to the emulator software and are only meaningful in describing a class of correct command entries. Items enclosed in brackets are optional parts of a command. The brackets themselves would never be entered on a command line, but the item within the brackets could optionally be included.

UPPERCASE — Characters shown in upper case must be entered exactly as shown. Uppercase is used to denote command keywords as shown in the following example:

***DEFINE** *label* = *expression* cr

class name — Italics denote general terms that must be replaced by a specific member of the class referenced. For example, *filename* would be replaced by a valid ISIS-iPDS filename and *address* would be replaced by a valid address.

[*option*] — Brackets enclose optional material that may or may not be included on the command line. For example, [=*address*] is an optional item that may be appended to the BREAK command.

. . . Ellipses indicate that the preceding item can be repeated. The following example indicates that more than one string or expression may be evaluated and displayed by using the WRITE command:

WRITE { *string* / *expr* } [*,string* / *,expr*] . . .

{ *item* / *item* } Braces indicate that one and only one of the enclosed entries must be selected. If the items are also enclosed by brackets, they are optional and no choice is required. For example:

{ Y / N }

Indicates a choice must be made to enter either Y or N. The following example indicates that either A or B or neither can be selected. The enclosed choices are printed in a vertical column.

[{ A / B }]

{ *item* / *item* } . . . Braces followed by ellipses indicate that at least one of the enclosed items must be selected. If the items are also enclosed by brackets, they are optional and no choice is required. The items may be used in any order unless otherwise specified. For example:

{ A / B / C / D } . . .

indicates that a choice must be made to include one or more of the items A, B, C, or D.

INCLUDE The underline denotes a valid abbreviation of the command word.

Command Description Formats

The EMV-51A commands appear in alphabetical order as a reference guide for the experienced user. Each command begins with the command keyword in color at the outside margin on the page. The format is followed by a brief description of the items required in the command and short examples are given as illustrations of the command in use. This format is shown in Figure 10-1.

Special Command Format Terms

In addition to the notational conventions, the command format descriptions in this manual contain general terms that are common to many commands:

command	*destination pn*	*source pn*
command list	*expression*	*string*
condition	*expr*	
destination	*source*	

Expression

An expression *expr1* is composed of numerals, keywords, and symbolic references. The expression is evaluated to a 16-bit number that replaces the original expression.

A numeral consists of one or more digits and, optionally a one character suffix that specifies the desired number base. The value can be preceded by a unary sign that indicates whether the value is positive or negative. Table 10-1 lists the number bases and the numerical digits recognized by the EMV-51A software.

COMMAND KEYWORD

Brief phrase describing command

Syntax

KEYWORD *paramter 1 through parameter n*

Where:

parameter 1	description
parameter n	description

Discussion

Comments related to operating the command.

Examples

A few brief illustrations of the command line.

Figure 10-1 Format of Command Descriptions

Table 10-1 Available Number Bases and Digits

Number Base	Valid Digits	Explicit Radix	Example
Binary (base 2)	0,1	Y	11110011Y
Octal (base 8)	0-7	Q	363Q
Decimal (base 10)	0-9	T	243T
Hexadecimal (base 16)	0-9, A-F	H	F3H

Examples:

123T	Decimal numeric expression
12H*23H	Hexadecimal numerals connected by an arithmetic operator to form an expression.
.ABLE AND .BAK	Symbolic references connected by a logical operator to form an expression.

String

The term string refers to a sequence of one or more alphanumeric or special characters enclosed in apostrophes (′), as follows:

′HELLO′	The apostrophes (′) enclose the string HELLO.

If a string contains an apostrophe, use two apostrophes to mark the apostrophe. When the string is displayed on the screen, the emulator software displays the double apostrophe as a single apostrophe. The following example contains an embedded apostrophe.

′THEY′′RE′	Double apostrophe marks an embedded apostrophe.

When a string of characters is stored in memory, each character occupies one byte or memory location. In the previous example, the string HELLO would occupy 5 memory locations. If the string has more than one character, the emulator uses consecutive memory locations to store each character.

Source

The term *source* in command lines refers to an assembler mnemonic input to the command. The *source pn* is the assembler mnemonic input file.

Destination

The term *destination* in command lines refers to the output for the command. The *destination pn* is the output file.

Command

The term *command* in command lines refers to any valid EMV-51A command. The *command list* is a sequence of EMV-51A commands.

Condition

A *condition* can have the same kinds of entries as *expr*. However, in a *condition* only the least significant bit (LSB) of the result is tested.

Alphabetical Listing of Commands

The following pages contain an alphabetical list of the EMV-51A commands and their command formats.

ACCUMULATOR

Display or change the contents of the accumulator

Syntax

ACC [=*expr*]

Where:

expr specifies the 8-bit numeric value that replaces the current contents of the accumulator. The value can be specified as a symbolic name or an expression.

Discussion

The accumulator (ACC) is a general purpose register used in data transfers and arithmetic operations. The ACC command displays or changes the contents of the register. When *expr* is used, it is evaluated to a 16-bit number and truncated to the lower 8-bits.

Examples

Key-in Sequence	Comments
ACC RETURN	Displays the current contents of the accumulator. The numeric value is displayed according to the default base.
ACC = 10H RETURN	Changes the contents of the accumulator to the value 10H.

ASM

Assembles one line of instruction mnemonics into program memory

Syntax

ASM [{ ORG *expr* / *source* }]

Where:

ORG *expr*	changes the contents of the assembly location counter to the 16-bit value of *expr*. The expression consists of numeric values and symbolic references.
source	specifies a valid instruction mnemonic.

Discussion

The ASM command displays the current value of the assembly location counter, changes the current contents to the indicated value, or assembles one microcontroller instruction into program memory.

The assembly location counter is a pointer that indicates where the next available location is in program memory. It is automatically incremented for each instruction stored in memory and the new value is displayed.

When a new instruction is assembled into program memory, it is stored at the location indicated by the contents of the assembly location counter. Refer to the *Microcontrollers Handbook*, order number 210918, for additional information on the MCS-51 microcontroller instruction set.

Examples

Key-in Sequence	Comments
ASM RETURN	Displays the current address value in the assembly location counter.
ASM ORG 0 RETURN	Sets the assembly location counter to program memory location 0.
A NOP RETURN	Assembles the NOP instruction into memory at the location pointed to by the assembly location counter.

B

Display or change the contents of the multiplication register

Syntax

B [=*expr*]

Where:

expr	specifies the 8-bit numeric value that replaces the current contents of the B register. The value can be specified as a symbolic name or an expression.

Discussion

The multiplication register (B) is used during arithmetic operations and as a general purpose register for all remaining operations. The B command displays or changes the contents of the register. When *expr* is used, it is evaluated to a 16-bit number and truncated to the lower 8 bits.

Examples

Key-in Sequence	Comments
B RETURN	Displays the current contents of the B register. The numeric value is displayed according to the default base.
B=10H RETURN	Changes the contents of the B register to the value 10H.

BASE

Changes the base for display of numeric data

Syntax

BASE [= (Y / H / T / Q)]

Where:

Y	specifies binary as the base for display of numeric data. This affects only the I/O port displays (BUS, P1, P2, and P4 through P7). The other values displayed retain the current base.
H	specifies hexadecimal as the base for display of numeric data. Hexadecimal is the initial default base for display.
T	specifies decimal as the base for display of numeric data.
Q	specifies octal as the base for display of numeric data.

Discussion

The BASE command displays or changes the base for the display of numeric values. The initial base is hexadecimal (H). When the current base is specified as binary, only the I/O port displays (BUS, P1, P2, and P4 through P7). All other numeric values are displayed according to the contents of BASE.

Examples

Key-in Sequence	Comments
BASE RETURN	Displays the base for numeric values displayed on the screen.
BAS = T RETURN	Changes the base to decimal.
BAS = Y RETURN	Changes the display base of the I/O ports only.

BC

Clears all breakpoints

Syntax

BC

Discussion

The BC command clears all set breakpoints. After the BC command is executed, breakpoints are not recognized during emulation until they are set again.

Examples

Key-in Sequence	Comments
BC	Removes all set breakpoints.

BRB

Enables or disables
the branch breakpoint

Syntax

BRB [= { ON / OFF }]

Discussion

The command, BRB, enables or disables the branch breakpoint. The emulator places a breakpoint at the location of the next instruction that causes a branch prior to the start of GO emulation and if BRB is enabled. The BRB breakpoint disables all other breakpoints while it is active. During STEP emulation, the BRB breakpoint is ignored. The default state of BRB is OFF.

The emulator does not distinguish between conditional branches that fail (fall through) and those that pass (branch). The break occurs on the fetch of the branch instruction, not on its outcome.

Examples

Key-in Sequence	Comments
BRB = ON RETURN	Enables the branch breakpoint.
BRB = OFF RETURN	Disables the branch breakpoint.

BREAK

Displays a table containing all break commands and their values

Syntax

BREAK

Discussion

The BREAK command displays a table containing all breakpoint commands and their current states. The display remains on the screen until a non-break command is entered, such as the GO command. When a break command is entered with a new state or content, the table is updated to reflect the new contents.

Examples

Key-in Sequence	Comments
BREAK RETURN	Displays a table consisting of all the break commands and their current state or content. The table remains on the screen until a non-break command is entered.

Figure 10-2 shows an example of the table displayed by the BREAK command.

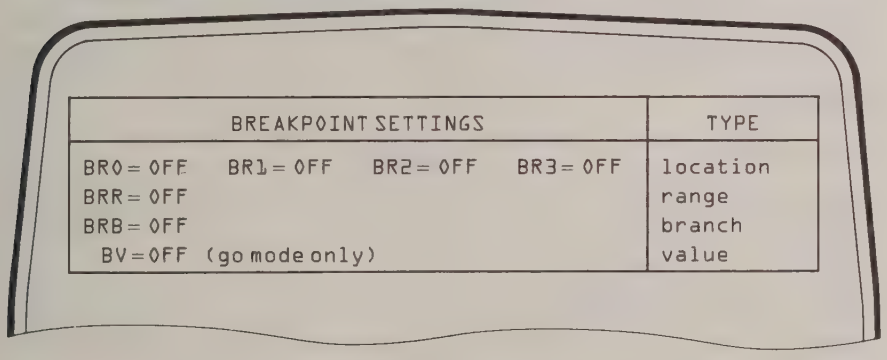


Figure 10-2 The BREAK Command Display Table

BRR

Specifies a range breakpoint

Syntax

BRR [= { *expr1* TO *expr2* / OFF }]

Where:

expr1	specifies the beginning address within the range of addresses. It can be specified as an actual address, a symbolic name, or an expression.
expr2	specifies the ending address within the range of addresses. It can be specified as an actual address, a symbolic name, or an expression.
OFF	disables the range breakpoint.

Discussion

The BRR command specifies a range breakpoint between the limits of *expr1* and *expr2*, where *expr1* is less than or equal to expr2. When breakpoints are enabled and program execution branches to an address within the specified range during an emulation, the EMV-51A hardware stops the current emulation and returns control to the user. The default state of BRR is OFF. No more than four breakpoints can be set at any one time (see the section Breakpoint Restrictions in Chapter 7 for details).

Examples

Key-in Sequence	Comments
BRR = 100H TO 150H RETURN	Sets a range breakpoint between addresses 100H and 150H.
BRR = .ABLE + 4 TO .SETUP − 2 RETURN	Sets a range breakpoint between the addresses indicated by the expressions .ABLE+4 and .SETUP−2.
BRR = OFF RETURN	Disables the range breakpoint.
BRR RETURN	Displays the current range setting.

BR*n*

Specifies up to four address breakpoints

Syntax

BR*n* [= { *expr* / OFF }]

Where:

n	specifies a numeric value between 0 and 3 that indicates the desired address breakpoint. Break 0 may be used also as an arming breakpoint.
expr	consists of numeric values, symbol names, and mathematical operators. The expression is evaluated to a 16-bit result.
OFF	disables the specified address breakpoint(s).

Discussion

The BR*n* command specifies from 1 to 4 address breakpoints: BR0, BR1, BR2, and BR3. Up to 4 address breakpoints can be specified.

When an instruction is fetched at an address specified by any address breakpoint during emulation, the emulation is stopped and control is returned to the user.

The default state of the address breakpoints is OFF.

Examples

Key-in Sequence	Comments
BR0 = 100H RETURN	Sets an address breakpoint for address 100H.
BR2 = .ABLE + 5 RETURN	Sets an address breakpoint for address specified by the expression .ABLE + 5.
BR3 = OFF RETURN	Disables address breakpoint number 3.
BR0 RETURN	Displays the current setting of breakpoint 0.

BV

Specifies a register value breakpoint

Syntax

BV [= { *register expr* | OFF }]

Where:

register	specifies one of the following registers of the 8051 microcontroller: R0-R7 or ACC.
expr	consists of numeric values, symbol names, and mathematical operators. The expression is evaluated to 8 or 16 bits.

Discussion

The BV command specifies a register and specific value as the value breakpoint. It is used in the STEP emulation mode. The *register* is one of the 8051 microcontroller registers, while the value is an 8- or 16-bit number depending on the register selected.

When an instruction is executed that references the specified register and the register contains the indicated value, program emulation is stopped and control is returned to the user.

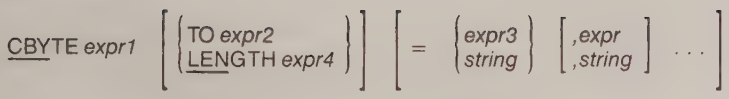
Examples

Key-in Sequence	Comments
BV = ACC 87H RETURN	Specifies the accumulator when it contains the value 87H as the value breakpoint. When the accumulator contains 87H during emulation, the emulator software stops program execution and returns control to the user.
BV = OFF RETURN	Disables the value breakpoint.

CBYTE

Display or modify the contents of program memory

Syntax



Where:

expr	is the beginning address in program memory.
expr1	specifies a beginning address within program memory. It can be a label or a numeric address.
TO *expr2*	specifies an ending address within program memory. It can be a label or a numeric address.
expr3	specifies the 8-bit value stored at the address specified in *expr1*. If a value greater than 0FFH is specified, the software uses the least significant byte as the result. The expression can consist of numerals, symbol names, and arithmetic operators.
string	specifies an alphanumeric string of characters that are stored in memory beginning at *expr1*. Each character in the string is stored in successive memory locations.
LENGTH *expr4*	specifies a range of locations within program memory to be displayed or operated on. The *expr4* indicates the number of locations to be displayed.

Discussion

The CBYTE command displays or modifies the contents of program memory. The command can display or modify the contents of EMV-51A program memory. The contents of external program memory can be read, but not modified. This external program memory will either be 4K to 64K or 8K to 64K, depending on the jumper configuration.

If the TO or LENGTH parameters are entered, the command displays the memory contents at the indicated addresses. If the equal parameter (=) is entered, the command modifies the contents of memory to the specified value. The optional TO, LENGTH, and equal sign can be combined within the command line to address multiple memory locations.

Examples

Key-in Sequence	Comments
CBYTE 0 TO 3FH RETURN	Displays the contents of program memory locations 0 to 3FH.
CBYTE 0 LEN 9 = 1,2,CBYTE 56H RETURN	Changes the contents of program memory locations 0 through 9 to the sequence 1, 2, and the contents of program memory location 56H. The sequence (1, 2, CBYTE 56H) is repeated until all 10 memory locations are changed.

CDUMP

Display the contents of program memory as both hexadecimal and ASCII values

Syntax

CDUMP *expr1* TO *expr2*

Where:

expr1	specifies the beginning address of a range of code addresses to be displayed.
TO *expr2*	specifies the ending address of a range of code addresses to be displayed.

Discussion

The CDUMP command displays the contents of program memory as hexadecimal values and ASCII characters. The hexadecimal values are displayed on the left side of the display, while the ASCII characters are displayed on the right side of the display.

Examples

Key-in Sequence	Comments
CDUMP 80H TO 83H RETURN	Displays the contents of program memory locations 80 to 83H as hexadecimal values and ASCII characters.
CDUMP .ABLE TO .FIN RETURN	Displays the contents of program memory as hexadecimal values and ASCII characters. The addresses are specified by the contents of the symbolic names .ABLE and .FIN.

COUNT

Begins a program loop

Syntax

```
COUNT expr <cr>
    [ UNTIL condition <cr>
      WHILE condition <cr>
      command <cr> ] ...
ENDCOUNT
```

Where:

expr	specifies the number of times the loop is repeated. This is evaluated to a decimal value unless an explicit base override is used.
WHILE	indicates that the loop terminates when the least significant bit of the condition tested is false.
UNTIL	indicates that the loop terminates when the least significant bit of the condition tested is true.
condition	indicates the test condition that satisfies the WHILE/UNTIL clauses. Only the least significant bit of the *condition* is tested.
command	is any command that executes until the test condition is met.

Discussion

The COUNT...END command sequence forms a bounded command loop that executes until a specified count is reached or a specified condition is encountered. If the optional test condition is met during the loop execution, the programming loop terminates before the final count is reached. The optional test value is entered with the WHILE or UNTIL clause.

Examples

Key-in Sequence

COUNT 5 RETURN

WHILE DBYTE 0 < .TOTAL
RETURN

STEP RETURN

DBYTE 0 = (DBYTE 0) + 1
RETURN

END RETURN

Comments

Executes the loop five times or until the contents of DBYTE 0 equals or exceeds the value assigned to the symbolic name .TOTAL.

```
COUNT = .CNT RETURN
STEP RETURN
UNTIL DBYTE 0 = .TOTAL RETURN
DBYTE 0 = (DBYTE 0) + 1 RETURN
END RETURN RETURN
```

Executes the loop 10 times or until the contents of DBYTE 0 equals the value assigned to the symbolic name .TOTAL.

DASM

Displays program memory as instruction mnemonics

Syntax

DASM *expr1* TO *expr2*

Where:

expr1	specifies the beginning address of the range of addresses to be disassembled. It consists of numeric values, symbolic references, and mathematical operators.
TO *expr2*	specifies the end address of a range of addresses to be disassembled. It consists of numeric values, symbolic references, and mathematical operators.

Discussion

The DASM command disassembles the contents of program memory and displays the contents as mnemonic instructions. If the symbolic display is enabled (via the ENABLE command) and a match is found, the DASM command displays opcode addresses as symbolic names.

An instruction is displayed if its first byte is within the range of indicated addresses. The address contained in *expr1* must point to the first byte of an instruction. Otherwise, the command produces invalid results.

Examples

Key-in Sequence	Comments
DASM 100H TO 150H RETURN	Disassembles the contents of program memory locations 100H through 150H.
DAS .ABLE TO .BAKER RETURN	Disassembles the contents of program memory beginning with the location indicated by the contents of the symbolic name .ABLE and ending with the location indicated by the contents of the symbolic name .BAKER.

DBYTE

Display or modify the contents of internal data memory

Syntax

DBYTE *expr1* [{ TO *expr2* / LENGTH *expr4* }] [= { *expr3* / *string* } [,*expr* / ,*string*] ...]

Where:

expr1	specifies a beginning address within internal data memory.
TO *expr2*	specifies an ending address within internal data memory. The address must be between 0 and 7FH.
LENGTH *expr4*	specifies a range of locations within internal data memory to be displayed. The *expr4* indicates the number of locations to be displayed. This ranges between 1 and 80H.
expr3	specifies the 8-bit value stored at the address specified in *expr1*. The value consists of numerals, symbol names, and arithmetic operators.
string	specifies an alphanumeric string of characters that are stored in memory beginning at *expr1*. Each character in the string is stored in successive memory locations.

Discussion

The DBYTE command displays or modifies the contents of internal data memory, depending on how the command is entered.

If the TO or LENGTH parameters are entered, the command displays the memory contents at the indicated address. If the equal parameter (=) is entered, the command modifies the contents of memory to the specified value. The optional TO, LENGTH, and equal sign can be combined within the command line to assign values to multiple memory locations.

Examples

Key-in Sequence	Comments
DBYTE 0 TO 3FH RETURN	Displays the contents of data memory locations 0 to 3FH.
DBYTE 0 LEN 9 = 56H RETURN	Changes the contents of data memory locations 0 through 8 to 56H.
DBYTE 7 = 'THIS IS A STRING' RETURN	Changes the contents of data memory beginning at address 07H and continuing for the length of the string.

DDUMP

Display contents of data memory as both hexadecimal and ASCII values

Syntax

DDUMP *expr1* TO *expr2*

Where:

expr1	specifies the beginning address of a range of data addresses to be displayed.
TO *expr2*	specifies the ending address of a range of data addresses to be displayed.

Discussion

The DDUMP command displays the contents of data memory as hexadecimal values and ASCII characters. The hexadecimal values are displayed on the left side of the display, while the ASCII characters are displayed on the right side of the display.

Examples

Key-in Sequence	Comments
DDUMP 8H TO 4AH RETURN	Displays the contents of data memory locations 8H to 4AH as hexadecimal and ASCII values.
DDUMP .ABLE + 4 TO .SEC + 10H RETURN	Displays the contents of data memory as hexadecimal values and ASCII characters.

DEFINE

Defines symbolic names or macros

Syntax

DEFINE { *:string* | *.label = expr* }

Where:

:string	specifies the beginning of a macro where *string* is an alphanumeric string of 2 to 32 characters, including the colon, that is the name of the new macro.
.label	specifies an alphanumeric string of 2 to 32 characters, including the period, that becomes a user symbolic name.
expr	specifies the 16-bit value to be assigned to the symbol name. It can consist of numeric values, symbolic references, and mathematical operators.

Discussion

The DEFINE command performs two tasks:

1. Evaluates the expression entered with it and assigns the 16-bit result to the symbolic name. Once assigned, the symbolic name rather than the numeric value or expression can be specified. The symbolic name is added to the end of the EMV-51A user symbol table. An error occurs if the entered symbol name already exists in the symbol table.

2. Begins a new macro where *:string* is the name of the new macro. All commands following the DEFINE command are placed in the new macro until the EM command. Once the macro definition is created, it can be saved oisk file with od (described later in this chapter).

Examples

Key-in Sequence	Comments
DEFINE .CHALK = .ABLE + FFH RETURN	Assigns the result of the expression .ABLE + FFH to the new symbolic name .CHALK.
DEF :NULL RETURN	Begins a macro definition labeled :NULL. All commands following the DEFINE command are entered into the macro until the end macro (EM) command. The macro name is added to a user macro list maintained by the emulator software.

DIR

Displays the names of all available macros

Syntax

DIR

Discussion

The DIR command displays the names of all macros in the user macro list. This includes macros in RAM memory and in the temporary macro workfiles. The macro text is not displayed.

DISABLE

Disables the display of
symbolic names or macro text

Syntax

DISABLE { SYMBOLIC / EXPANSION }

Discussion

The DISABLE command performs two functions:

1. Disables the display of symbolic names for the screen display commands: DTRACE, ASM, and the entry or display of memory locations. When a numeric value is encountered in one of the commands, the value itself is displayed according to the current BASE.

2. Disables the expansion of macros prior to the execution of the macro. The default condition is the disable condition, where macros are not expanded.

Examples

Key-in Sequence	Comments
DIS SYM RETURN	Disables the use of symbolic names for numeric values.
DIS EXP RETURN	Disables the expansion of macros prior to execution of the macro.

DPTR

Display or change the contents of the data point register

Syntax

DPTR [=*expr*]

Where:

expr specifies the 16-bit address value to be assigned to the data pointer register. It can consist of numeric values, symbol names, and mathematical operators.

Discussion

The data pointer register (DPTR) serves as the base register in indirect jumps, table look-up instructions, and external data transfers. The DPTR command displays or changes the contents of the data pointer register. When *expr* is used, it is evaluated to a 16-bit value.

Examples

Key-in Sequence	Comments
DPTR RETURN	Displays the current contents of the data pointer register. The numeric value is displayed according to the default base.
DPTR = 100H RETURN	Changes the contents of the data pointer register to the value 100H.

DTRACE

Displays a table containing all trace display commands

Syntax

DTRACE

Discussion

The DTRACE command displays a table containing the trace display commands and their current condition, whether enabled or disabled. The table remains on the screen until a non-trace display command is entered, such as the GO command. When a trace display command is entered with a new state or content, the table is updated to reflect the new contents.

Examples

Key-in Sequence	Comments
DTRACE RETURN	Displays a table consisting of the trace display commands and their current state or content. The table remains on the screen until a non-trace command is entered.

ENABLE

Enables the disolay of symbolic names or macro text

Syntax

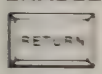
Discussion

The ENABLE command performs two functions:

1. The display of symbolic names for the screen display commands DTRACE, ASM, and the entry or display of memory locations. When a numeric value corresponds with a value assigned to a symbolic name, the symbolic name is displayed rather than the numeric value. Otherwise, numeric values are displayed as a symbolic name plus an offset.

2. Enables the expansion of macros prior to the execution of the macro definition. The default state is macro expansion disabled.

Examples

Key-in Sequence	Comments
ENABLE SYMBOLIC 	Enables the use of symbolic names for numeric values.
ENA EXP	Enables the expansion of macros prior to the execution of the macro.

EVALUATE

Evaluates an expression

Syntax

EVALUATE *expr*

Where:

expr	specifies an arithmetic expression or symbolic name that consists of numeric values, symbolic references, and mathematical operators. The expression is evaluated and a 16-bit result is generated.

Discussion

The *expr* is evaluated and a 16-bit numeric value is produced. The value is displayed in binary, octal, decimal, hexadecimal, and the ASCII equivalent. In addition, the value is displayed as an offset from a symbol name that is less than the value.

Examples

Key-in Sequence	Comments
EVA 30H+23*(.ABLE+4) RETURN	Evaluates the *expr* 30H+23*(.ABLE+4) and displays the 16-bit result in four bases, ASCII, and an offset from the nearest symbol name.
EVALUATE .ABLE RETURN	Displays the value assigned to the symbolic name .ABLE and displays the result in four bases, ASCII, and an offset from the nearest symbol name.

EXIT

Terminates the debugging session
and returns control to the ISIS-iPDS operating system

Syntax

EXIT

Discussion

The EXIT command terminates the debugging session and returns control to the ISIS-iPDS operating system or to the monitor if the disk in drive 0 is not a system disk. The command performs a hardware reset before exiting. The user's program and macros should be saved if desired prior to entering the EXIT command.

Example

Key-in Sequence	Comments
EXIT	Terminates the current debugging session and returns control to the operating system.

FUNCTION

Displays or assigns function keys

Syntax

FUNCTION [= *n* :*string*]

Where:

n	is a numeric value between 0 and 9 inclusive
string	is a pre-defined macro name, where the name consists of 2 to 9 alpha-numeric characters including the colon.

Discussion

The FUNCTION command assigns macro names to the specified function keys. Do not assign macros requiring parametes to function keys. Function keys 1 through 4 are pre-defined as follows:

Executes the MEMORY command and produces the tabl containing the display/modify commands.

Executs the BREAK command and produces the table containing the trace display commands.

Executes the DTRACE command and produces the table containing the trace display commands.

Executes the REGISTER command and produces the table containing the register control commands.

The function keys 1 through 4 can be re-defined, but special steps must be taken to retrieve the original definitions. The two methods for retrieving the original definitions are as follows:

1. EXIT the emulation session and reload the EMV-51A software by entering the EMV51A command. This procedure retrieves the original function key definitions.

2. Define a macro that duplicates the original function and assign the macro to the desired function key.

Perform the following steps to incorporate function keys in command files:

1. Enter a numeric value within the range of 0 through 9. This value corresponds to the desired function key number.

2. Enter a carriage return.

When the command file is loaded with the INCLUDE command, the numeric value between the carriage returns executes as if the FUNCT key and one of the numeric keys 0-9 on the keyboard had been pressed.

Examples

Key-in Sequence	Comments
FUNCTION = 0 :DEMO	Assigns the macro :DEMO to function key 0.
FUN = 5 :MULT	Assigns the macro :MULT to function key 5.
FUN	Displays all function key assignments.

NOTE

All predefined and user defined functions can be accessed by pressing the number associated with the function without pressing the FUNCT key.

GO

Begins full speed emulation of the user's program

Syntax

GO [[FROM] *expr*]

Where:

expr specifies an address where emulation is to begin. The *expr* consists of numeric values, symbolic references, and mathematical operators. The expression is evaluated to a 16-bit result.

Discussion

The GO command begins program emulation from the address contained in the program counter or from the address entered with the command. The GO command differs from the STEP command as follows:

1. Breakpoints are checked in hardware with the GO command (they are checked in software with the STEP command).

2. Trace display controls are not checked during execution of the GO command.

Examples

Key-in Sequence	Comments
GO RETURN	Begins emulation at the address contained in the program counter and proceeds indefinitely or until a breakpoint is encountered.
G FROM 100H RETURN	Begins emulation at address 100H and proceeds indefinitely or until a breakpoint is encountered.

HELP

Displays general or specific information about EMV-51A commands

Syntax

HELP [*item*][,...]

Where:

item specifies an EMV-51A command or keyword.

Discussion

The HELP command provides information about EMV-51A commands and keywords. Multiple items can be entered with the command.

Examples

Key-in Sequence	Comments
HELP RETURN	Displays a list of the HELP items. Assistance can be obtained with any item in the list.
HELP LOAD,LIST RETURN	Displays information about the LOAD and LIST commands.

IF...THEN...ELSE

Conditionally executes
a series of commands

Syntax

```
IF cond1 [THEN] <cr>
    [command <cr>. . .]
  [ ORIF cond2 <cr>
      [command <cr>]. . . ]  . . .
  [ ELSE <cr>
    [command <cr>]. . . ]
ENDIF
```

Where:

cond1	specifies a conditional test whose 16-bit result is true or false. Consists of numeric values, symbolic references, and mathematical operators.
cond2	specifies a conditional test whose result is true or false. Consists of numeric values, symbolic references, and mathematical operators.
command	specifies one or a list of EMV-51A commands to be executed if the test condition is true.

Comments

The IF..THEN..ELSE conditional command sequence performs conditional execution of command sequences. The *cond1* and *cond2* specify a conditional tests whose 16-bit results are true (low-order bit is a one) or false (low-order bit is a zero). If the result is true, the command(s) following the conditional test are executed. Otherwise, the ELSE or ENDIF commands are executed.

During command execution, the emulator software performs the following tests on the IF..THEN..ELSE sequence:

1. If *cond1* is true, execute the *command* following it and branch to ENDIF.
2. If *cond1* is false, test *cond2*. If true, execute the command list following *cond2* and branch to ENDIF. If multiple ORIF's are present, test each *cond2* until one is true or until all test false.
3. If *cond1* and *cond2* are both false, execute the *command* following the ELSE and branch to ENDIF.

The *command* specifies one or more of the EMV-51A commands. The series of commands are executed and then control branches to the ENDIF command.

Examples

Key-in Sequence	Comments
IF DBY 100 = 1 RETURN	This performs the conditional test DBYTE 100 = 1. If the test is true, the command STEP

STEP FROM 100 RETURN

ELSE RETURN

CBY 0 TO 200 RETURN

ENDIF RETURN

FROM 100 is executed and control passes to the ENDIF command. Otherwise the command CBYTE 0 TO 200 is executed.

IF .G=0 THEN RETURN

STEP FROM 100H RETURN

ORIF .G=1 THEN RETURN

.ABLE=.ABLE+1 RETURN

STEP FROM 200H RETURN

ORIF .G=.ABSCAN THEN RETURN

CBY 0 TO 300H RETURN

.G=.G+1 RETURN

ELSE RETURN

GO FROM 50H RETURN

ENDIF RETURN

The IF and ORIF commands perform conditional command execution by testing for a specified condition. Depending on the test result, the emulator software performs from 1 to 3 conditional tests. If all the tests fail, the command following the ELSE command is executed. If one of the tests passes, the commands following the test are executed, and control passes to the ENDIF command.

IF .T<3 THEN RETURN

.ABLE=10*.T RETURN

.T=.ABLE/10 RETURN

IF .T>1 THEN RETURN

The value of the symbolic lable .T determines the execution sequence of a series of nested IF commands. If the initial value of .T is less than 3, the testing begins. If the value of .T is greater than 1, the second IF command is executed. f the value of .T equals 1, the third IF command is executed.

.BAKER = .ABLE*.T/4H

.T = .BAKER/.ABLE

STEP FROM .T COUNT = .ABLE

IF .T = 1 THEN

STEP FROM .T COUNT = .BAKER

END

END

ELSE

STEP FROM .T BR

ENDIF

INCLUDE

Loads a macro or a command file

Syntax

INCLUDE *source pn*

Where:

source pn specifies the file name of the macro or command file.

Discussion

The *source pn* is any valid ISIS-iPDS device name followed by a filename as described in the *iPDS™ User's Guide*. The emulator software executes any valid EMV-51A commands as they are loaded from the *source pn*.

Macros are loaded with the INCLUDE command. Since the first command in the macro is the DEFINE command, the emulator software recreates the macro and lists the macro text on the screen. The DISABLE EXPANSION command has no effect when loading macros from a file.

Command files are also loaded with the INCLUDE command. A command file is created prior to loading the EMV-51A software and consists of valid EMV-51A commands. Refer to the *iPDS™ User's Guide* for additional information on command files.

The INCLUDE command can be nested within any other command, including the DEFINE command.

Examples

Key-in Sequence	Comments
INCLUDE :F1:MAC.SAV RETURN	Loads the contents of filename MAC.SAV on device 1, and executes the contents as they are entered. In this case the file consists of macros, but the file could be a command file.
INCLUDE :CI: RETURN	Loads the contents of filename :CI:. Since this file is the keyboard, EMV-51A commands can be entered as if they originated from a command file. A CRTL-Z must be entered to close the file.

INTERRUPT

Displays a table containing the interrupt indicators

Syntax

INTERRUPT

Discussion

The INTERRUPT command displays a table containing the interrupt enable (IE) and interrupt priority (IP) registers, including the status of interrupts in progress for priority 0 (IIP0) and priority 1 (IIP1). For IP, 0 and 1 indicate the priority levels with level 1 priority being higher than level 0. A 1 indicates an interrupt is enabled for IE, while a 0 indicates an interrupt in progress for both IIP0 and IIP1.

Example

Key-in Sequence	Comments
INTERRUPT RETURN	Displays the interrupt table as described in the discussion section.

LIST

Sends a copy of the debugging session to a disk file

Syntax

LIST *destination pn*

Where:

destination pn specifies the output file that receives a copy of the debugging session.

Discussion

The *destination pn* is any valid ISIS-iPDS device name followed by a filename as described in the *iPDS™ User's Guide*. The emulator software sends a copy of all input and output from the debugging session to the file.

The LIST command destroys any data that was initially in the file specified with the command. The command is terminated by entering the following command:

LIST :CO:

Examples

Key-in Sequence	Comments
LIST :F2:TEST.V3 RETURN	Saves a copy of all input and output from the debugging session to the file TEST.V3 until the LIST :CO: command is entered.
LIST :LP: RETURN	Saves a copy of all input and output from the debugging session to the line printer until the LIST :CO: command is entered.
LIST :CO: RETURN	Terminates the LIST command and closes the *source pn*.

LOAD

Loads the user's object code
or the symbol table or both

Syntax

LOAD *source pn* [NOCODE] [NOSYMBOL]

Where:

source pn	specifies the input file that contains the object code and symbol table of the user's program.
NOCODE	specifies the user's symbol table is loaded from the input file, but not the object code.
NOSYMBOL	specifies the user's object code is loaded from the input file, but not the symbol table.

Discussion

Depending on the options selected with the command, the LOAD command loads the user's object code program and symbol table from the input file. Unless otherwise specified, the command defaults to loading the object code and the symbol table. The command operates on EMV program memory.

The LOAD command loads non-relocateable (absolute) code in hex or object format. Thus, absolute assembler output or linker output can be loaded.

If the operation is not successful, an error message is displayed on the console.

Examples

Key-in Sequence	Comments
LOAD :F2:EMVDEM.OBJ RETURN	Loads the object code program and the symbol table from the file :F2:EMVDEM.OBJ.
LOAD EMVDEM.OBJ NOCODE RETURN	Loads only the user's symbol table from the file EMVDEM.OBJ.
LOAD EMVDEM.OBJ NOSYMBOL RETURN	Loads only the object code program without the symbol table from the file EMVDEM.OBJ.

NOTE

The LOAD command does not check for duplicate symbols when loading the symbol table. If the same program is loaded twice, all symbol names are duplicated in the user symbol table.

MACRO

Displays the text of one or more macros

Syntax

MACRO [;*string1* [,:*string2*,...]]

Where:

:*string*	is a predefined macro name that consists of 1 to 32 alphanumeric characters including the colon.

Discussion

If the MACRO command is entered by itself, the emulator software displays the text of all user defined macros. Otherwise, the command displays the text of all macros entered with the command.

Examples

Key-in Sequence	Comments
MACRO RETURN	Displays the text of all user defined macros.
MACRO :SUM,:DIV RETURN	Displays the text of the macros :SUM and :DIV.

MEMORY

Displays format requirements for display/modify commands

Syntax

MEMORY

Discussion

The MEMORY command displays a table consisting of the display/modify commands (CBYTE, DBYTE, PBYTE, CDUMP, DDUMP, and PDUMP) and their format requirements. The table is displayed on the screen until any command is entered. When a command is entered, the emulator software displays the results of the command, but does not preserve the MEMORY table.

Examples

Key-in Sequence	Comments
MEMORY RETURN	Displays a table containing the display/modify commands.
M RETURN	Displays a table containing the display/modify commands.

P

Displays the last two instructions emulated

Syntax

P

Discussion

The P command displays the previous two instructions emulated with either the GO or STEP commands. The command displays NO DATA if no instructions have been emulated or if Warning D0 has occurred.

Example

Key-in Sequence	Comments
	Displays the last two 8051 instructions emulated with either the STEP or GO commands.

PBYTE

Display or modify the contents of external data memory

Syntax

Where:

expr1	specifies a beginning address within external data memory.
TO *expr2*	specifies an ending address within external data memory. The address must be between 0 and 64K.
LENGTH *expr4*	specifies a range of locations within external data memory to be displayed. The *expr4* indicates the number of locations to be displayed. It ranges between 0 and 64K.
expr3	specifies the 8-bit value that will be stored at the address specified in *expr1*. The expression consists of numerals, symbol names, and arithmetic operators.
string	specifies an alphanumeric string of characters that are stored in memory beginning at *expr1*. Each character in the string is stored in successive memory locations.

Discussion

The PBYTE command displays or modifies the contents of external data memory, depending on how the command is entered.

If the TO or LENGTH parameters are entered, the command displays the memory contents at the indicated address. If the equal parameter (=) is entered, the command modifies the contents of memory to the specified value. The optional TO, LENGTH, and equal sign can be combined within the command line to assign values to multiple memory locations.

Examples

Key-in Sequence	Comments
PBYTE 0 TO 3FH RETURN	Displays the contents of external data memory locations 0 to 3FH.
PBYTE 0 LEN 9 = 56H RETURN	Changes the contents of external data memory locations 0 through 8 to the value 56H.
PBYTE 0 LEN 1K = 1,2,3,4 RETURN	Stores a repeating sequence in external data memory beginning at location 0 and continuing for the next 1024 locations.

PC

Display or change the contents of the program counter

Syntax

PC [=*expr*]

Where:

expr specifies a 16-bit memory address that replaces the current contents of the program counter. It consists of numeric values, symbolic references, and mathematical operators.

Discussion

The program counter (PC) is a 12 bit register that indicates the current execution address. The PC command displays or changes the contents of the register. When *expr* is used, it is evaluated to a 16-bit number.

Examples

Key-in Sequence	Comments
PC RETURN	Displays the current contents of the program counter. The numeric value is displayed according to the default base.
PC = 100H RETURN	Changes the contents of the program counter to the value 100H.

PSW

Display or change the contents of
the program status word register (PSW)

Syntax

PSW [= *expr*]

Where:

expr specifies the numeric value that replaces the current contents of the PSW. Consists of numeric values, symbolic references, and mathmatical operators. The expression is evaluated to a 16-bit result and truncated to the lower 8 bits.

Discussion

The program status word register contains 8 flags that indicate the status of the microcontroller. These flags are listed below corresponding to the bit position with the PSW.

bit 7	Carry flag (CY)
bit 6	Auxiliary carry flag (AC)
bit 5	User flag 0 (F0)
bit 4	Register bank select 1 (RBS1)
bit 3	Register bank select 0 (RBS0)
bit 2	Overflow flag (OV)
bit 1	Reserved flag not accessible to the ser (UTL)
bit 0	Parity flag (P)

The PSW command displays or changes the contents of the PSW. When the contents are being changed, the value is entered as a 8-bit binary value, a byte value, or an expression.

Examples

Key-in Sequence	Commens
PSW RETURN	Displays the contents of the 8051 PSW. The numeric value is displayed as a binary value.
PSW = 1 RETURN	Sets the parity flag and clears the remaining flags.

PUT

Saves the definition of user macros in a file

Syntax

PUT *destination pn* { ;*string*[,:*string*,...] / MACRO }

Where:

destination pn	specifies the output file that receives a copy of the macro.
string	is a pre-defined macro name, where the name consists of 2 to 32 alphanumeric characters including the colon.
MACRO	specifies all macros currently residing in RAM or workfiles.

Discussion

The PUT command saves macro definitions on the *destination pn*. From one to all macros currently residing in RAM or macro workfiles can be specified. The *destination pn* can be any valid ISIS-iPDS device attached to the iPDS unit.

If the specified macro name does not exist, it is created by the PUT command.

Examples

Key-in Sequence	Comments
PUT MAC.SAV :DIV,:SUM RETURN	Saves the macros :DIV and :SUM on the file MAC.SAV.
PUT MAC.SAV MACRO RETURN	Saves all macros currently in RAM on the file MAC.SAV.
PUT :LP: MACRO RETURN	Sends a copy of all macros currently in RAM or in the macro workfiles to the line printer.

RBIT

Display or modify the contents of bit-addressable memory

Syntax

RBIT *expr1* [{ TO *expr2* / LENGTH *expr4* }] [= { *expr3* / *string* } [,*expr* / ,*string*] ...]

Where:

expr1	specifies a beginning address in bit-addressable data memory that is between 0H and 0FFH.
TO *expr2*	specifies an ending address in bit-addressable data memory. The address must be between 0H and 0FFH.
LEN *expr4*	specifies a count of locations in bit-addressable data memory to be displayed. The *expr4* indicates the number of locations to be displayed. It ranges between 0 and FFH.
expr3	specifies the 1-bit value that is stored at the specified address. The expression consists of numerals, symbol names, and arithmetic operators.
string	specifies an alphanumeric string of characters. Each character's least significant bit is stored in successive memory locations.

Discussion

The RBIT command displays or modifies the contents of bit-addressable data memory, depending on how the command is entered.

If the TO or LEN paameters are entered, the command displays the memory contents at the indicated address. If the equal parameter (=) is entered, the command modifies the contents of memory to the specified value. The optional TO, LEN, and equal sign can be combined within the command line to assign values to multiple memory locations.

Examples

Key-in Sequence	Comments
RBIT 20H TO 27H RETURN	Displays the contents of bit-addressable memory locations 20 to 27H.
RBIT D0H TO D7H = 1 RETURN	Changes the contents of the program status word register (PSW) to the value FFH.

RBS

Display or change the contents of register bank select flags

Syntax

RBS [=*expr*]

Where:

expr specifies the numeric value that replaces the current contents of register bank select flags in the program status register. The 2-bit value can be specified as a physical address, symbolic name, or an expression with a value between 0 and 3, inclusive.

Discussion

The microcontroller contains four banks of 8 general purpose registers numbered R0 through R7. The RBS command specifies one of the four banks of general purpose registers that is the active register bank. When *expr* is used, it is evaluated to a 2-bit value between the limits of 0 to 3, inclusive.

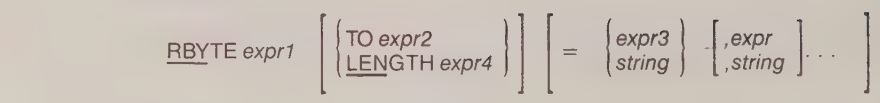
Examples

Key-in Sequence	Comments
RBS RETURN	Displays the current contents of the RBS flags in the program status register. The numeric value is displayed according to the current base.
RBS=1 RETURN	Changes the contents of the RBS flags in the program status register to the value 01, indicating general purpose register bank 1.

RBYTE

Display or modify the contents
of special function register memory

Syntax



Where:

expr1	specifies a beginning address within register memory.
TO *expr2*	specifies an ending address within register memory. The address must be between 80H and FFH.
LEN *expr4*	specifies a range of locations within register memory to be displayed. The *expr4* indicates the number of locations to be displayed. It ranges between 0 and 7FH.
expr3	specifies the 8-bit value that is stored at the specified address. The value consists of numerals, symbol names, and arithmetic operators.
string	specifies an alphanumeric string of characters that are stored in memory beginning at *expr1*. Each character in the string is stored in successive memory locations.

Discussion

The RBYTE command displays or modifies the contents of register memory, depending on how the command is entered.

If the TO or LEN parameters are entered, the command displays the memory contents at the indicated address. If the equal parameter (=) is entered, the command modifies the contents of memory to the specified value. The optional TO, LEN, and equal sign can be combined within the command line to assign values to multiple memory locations.

Examples

Key-in Sequence	Comments
RBYTE 80H TO 83H RETURN	Displays the contents of register memory locations 80 to 83H.
RBYTE 84H LEN 4 = 2 RETURN	Changes the contents of register memory locations 84H through 87H to the value 2.

REGISTER

Displays the 8051 registers and their contents

Syntax

REGISTER

Discussion

The REGISTER command displays a table consisting of the 8051 registers and their contents. The display remains on the screen until a non-register command is entered. When a register command is entered, the table is updated to reflect the new contents entered with the command.

Examples

Key-in Sequence	Comments
REG RETURN	Displays a table containing the registers and their contents.

Rn

Display or change the contents of the general purpose registers

Syntax

R*n* [= *expr*]

Where:

n	signifies a digit with the limits of 0 to 7 inclusive, that specifies one of the general purpose registers R0 through R7 within the register bank selected via RBS.
expr	specifies the numeric value that replaces the current contents of the general purpose register. The 8-bit value can be specified as a physical address, symbolic name, or an expression.

Discussion

The microcontroller contains four banks of 8 general purpose registers numbered R0 through R7. Depending on the particular bank selected via the RBS command, the R*n* command displays or modifies the contents of the register. When *expr* is used, it is evaluated to a 16-bit value and truncated to 8 bits.

Examples

Key-in Sequence	Comments
R0 RETURN	Displays the current contents of general purpose register R0. The numeric value is displayed according to the default base.
R2 = 10H RETURN	Changes the contents of general purpose register R2 to the value 10H.

REMOVE

Removes user-symbolic names or macro definitions

Syntax

REMOVE { *.string*[,*.string*,...] | *:string* [,*:string*,...] | SYMBOLS | MACRO }

Where:

.string	is the name of user-defined symbol(s) in the symbol table that are 2 to 32 characters long including the period.
:string	is the name of user-defined macro(s) in RAM or the workfiles that are 2 to 32 characters long including the colon.
SYMBOLS	specifies that all user symbolic names are removed from the user symbol table.
MACRO	specifies that all macros are removed from RAM or the workfiles.

Discussion

The REMOVE command deletes from one to all user symbolic names from the symbol table, or one to all macros from RAM or the workfiles.

Examples

Key-in Sequence	Comments
REMOVE .ABLE,.BAKER RETURN	Removes the user symbolic names .ABLE and .BAKER.
REMOVE SYMBOLS RETURN	Removes all user assigned symbolic names.
REMOVE :DIV RETURN	Removes the macro :DIV.
REMOVE MACRO RETURN	Removes all user macros from memory.

REPEAT

Begins a command loop

Syntax

```
REPEAT <cr>
   [ WHILE condition <cr>
     UNTIL condition <cr>   ...
     command <cr> ]
END
```

Where:

WHILE	indicates that the *command list* executes as long as the *condition* entered with the WHILE command is true.
UNTIL	indicates that the *command list* executes as long as the *condition* entered with the UNTIL command is false.
condition	indicates the test condition that satisfies the WHILE/UNTIL commands. The test condition consists of an arithmetic, logical, or relational test, such as the following: **DBYTE 0 < .TOTAL**
command	is a sequence of EMV-51A commands that execute until the test condition is met.

Discussion

The REPEAT..END command sequence sets up a command loop that executes until the optional specific test conditions are met. The ESC key can be used to exit from REPEAT loops.

The REPEAT..END command sequence with the WHILE clause, executes a series of EMV-51A commands as long as a test condition is true. The REPEAT..END command sequence with the UNTIL clause executes a series of EMV-51A commands as long as a test condition is false. Both command loops permit the user's sequence of commands to control the number of times the command loop executes. The REPEAT..WHILE and REPEAT..UNTIL command sequence can appear anywhere in the loop.

Examples

Key-in Sequence	Comments
REPEAT RETURN	The commands execute while the contents of DBYTE 0 equals or exceeds the value assigned to the symbolic name .TOTAL.
WHILE DBYTE 0 < .TOTAL RETURN	
ASM MOV.P0, # (DBY 0) RETURN	

```
DBY 0 = (DBY 0 + 1)
RETURN

END RETURN

REPEAT RETURN

UNTIL DBYTE 0 = .TOTAL
RETURN

ASM MOV.P0, # (DBY 0)
RETURN

DBY 0 = (DBY 0 + 1)
RETURN

END RETURN
```

The commands execute until the contents of DBYTE 0 equals the value assigned to the symbolic name .TOTAL.

RESET

Resets the emulator to its default settings

Syntax

RESET

Discussion

The RESET command initilizes the 8051 registers to zero, and sets all internal conditions to OFF.

The command sets the DTRACE and BREAK commands to their default states. The RESET command does not affect the SUFFIX and BASE commands, or the function key assignments.

Examples

Key-in Sequence	Comments
RES RETURN	Re-initializes the EMV-51A hardware and sets the DTRACE and BREAK commands to their default states.

SAVE

Saves the user's program in a file

Syntax

SAVE *destination pn* [*expr1* TO *expr2* | NOCODE] [NOSYMBOL]

Where:

destination pn	specifies the output file that receives a copy of the program.
expr1 TO *expr2*	indicates a specific range of the program that is saved in the output file.
NOCODE	specifies that the program code is not saved.
NOSYMBOL	specifies that the symbol table is not saved.

Discussion

Depending on the options selected with the command, the SAVE command puts a copy of the user's program and symbol table in the output file. The command options default to saving the program code and symbol table.

If a program range is not specified, the emulator software saves the entire 4K of EMV-51A program memory as the user's program.

If the operation is not successful, an error message is displayed on the console.

Examples

Key-in Sequence	Comments
SAVE :F1:EMVDEM.SAV RETURN	Saves a copy of the user's program and symbol table in filename EMVDEM.SAV on drive :F1:.
SAVE EMVDEM.S3 100H TO 1FFH RETURN	Saves the range of addresses 100H to 1FFH of the user's program and the symbol table in filename EMVDEM.S3 on drive :F0:.
SAVE :F2:EMVDEM.S2 NOCODE RETURN	Saves the user's program symbol table in filename EMVDEM.S2 on drive :F2:.
SAVE :F4:EMVDEM.S4 NOSYMBOL RETURN	Saves the user's program without the symbol table in filename EMVDEM.S4 on drive :F4:.

SP

Display or change the contents of the stack pointer

Syntax

SP [=*expr*]

Where:

expr specifies the memory location that replaces the current contents of the stack pointer. The location can be specified as a physical address, a symbolic name, or an expression that is evaluated to an 8-bit result.

Discussion

The stack pointer (SP) is a 3-bit register that indicates the current stack location. The SP command displays or changes the contents of the register. When *expr* is used, it is evaluated to an 8-bit value.

Examples

Key-in Sequence	Comments
SP RETURN	Displays the current contents of the stack pointer register. The numeric value is displayed according to the default base.
SP = 70H RETURN	Changes the contents of the stack pointer register to the value 70H.

STEP

Begins step emulation of the user's program

Syntax

STEP [[FROM] *expr*] [{ COUNT = *expr1* / BR }]

Where:

FROM *expr*	specifies an optional starting address within the user's program. The *expr* consists of numeric values, symbolic names, and mathematical operators that are evaluated to a 16-bit result.
COUNT = *expr1*	specifies an optional number of instructions to be emulated.
expr1	is a numeric value between 0 and FFFFH inclusive.
BR	enables software breakpoint checking.

Discussion

The STEP command emulates the user's program in a slow-down mode. During STEP emulation, the EMV-51A software emulates one instruction and then checks for the following items:

1. Trace display commands enabled. If trace display commands are set, the emulator software displays from one to four lines of trace information about the instruction emulated.

2. If BR is specified when the STEP command is entered, the emulator software performs software breakpoint checking. If a breakpoint is encountered, the software stops the emulation and returns control to the user.

NOTE

If a trace display command (TR*n* or TV) has turned off the software trace display, it remains off until another trace display command which is set to turn on the software trace display is executed, or until the EMV-51A software is reset.

Emulation begins with the instruction pointed to by the program counter or the address specified in the STEP command (the [FROM] *expr* option).

The COUNT = *n* option permits a range of instructions to be executed. If the COUNT = *n* option and the BR option are both specified, the BR option takes precedence over the COUNT option.

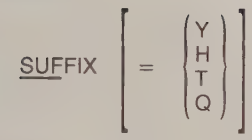
Examples

Key-in Sequence	Comments
STEP	Emulates one instruction of the user's program from the address contained in the program counter. The emulator software checks for trace display commands after each emulated instruction.
STEP FROM 100 BR	Begins emulation of the user's program from address 100H and enables software breakpoint checking. The emulator software checks for trace display commands after emulating each instruction and then checks for breakpoints.
STEP FROM 100H COUNT = 10	Emulates 10 instructions of the user's program, checking for trace display commands after each emulated instruction. The emulator software does not check for breakpoints.

SUFFIX

Displays or changes the default base for input numeric data

Syntax



Where:

Y	specifies binary as the default suffix for input numeric data.
H	specifies hexadecimal as the default suffix for input numeric data.
T	specifies decimal as the default suffix for input numeric data.
Q	specifies octal as the default suffix for input numeric data.

Discussion

The SUFFIX command displays or changes the default suffix for numeric values. The default suffix is overridden when an explicit suffix is entered with a numeric value. The initial default suffix is hexadecimal (H).

Examples

Key-in Sequence	Comments
SUFFIX RETURN	Displays the default suffix for numeric values.
SUF = T RETURN	Changes the default suffix to decimal. Unless an explicit suffix is entered with a numeric value, all numeric values are evaluated as base 10 (decimal).

SYMBOLS

Displays all user-assigned symbolic names and values

Syntax

SYMBOLS

Discussion

The SYMBOLS command displays all user-assigned symbolic names and their values. The command does not display system symbolic names.

Examples

Key-in Sequence	Comments
SYMBOLS RETURN	Displays all user-assigned symbolic names and their values.
SYM RETURN	Displays all user-assigned symbolic names and their values.

TB*n*

Enables or disables the display of up to four one-bit memory locations

Syntax

TB*n* [= { ON / OFF }]

Where:

n specifies one of four trace bit commands (0 through 3).

Discussion

The trace bit command (TB*n*) enables or disables the display of up to four one-bit memory locations within the direct bit-addressable memory space when emulation halts. Bit-addressable memory contains 128 user-defined software flags located between internal data memory address 20H through 2FH as well as several bit locations within several special function registers.

Examples

Key-in Sequence	Comments
TB0 RETURN	Displays the value for trace bit 0.
TB0 = 4H RETURN	Displays the contents of the fifth bit in bit-addressable memory as bit display TB0 when a breakpoint occurs or during STEP emulation mode.
TB1 = OFF RETURN	Disables the display of bit display TB1.

TD

Enables or disables the display of the disassembled instruction

Syntax

TD [= { ON / OFF }]

Discussion

The instruction trace (TD) command enables or disables the display of the disassembled instruction at a breakpoint or during STEP emulation mode. If enabled, the instruction at the breakpoint is disassembled and displayed, along with the cause of the breakpoint, on one line of the display.

Examples

Key-in Sequence	Comments
TD RETURN	Displays the current condition of the disassembled instruction function.
TD=ON RETURN	Enables the display of disassembled instructions and the cause of the breakpoint.

TM*n*

Display or change the contents of the timer/counter registers

Syntax

TM*n* [= *expr*]

Where:

n	specifies timer/counter register 0 or timer/counter register 1. The current value of the specified timer/counter register can be displayed or changed.
expr	specifies the 16-bit numeric value that replaces the current contents of the timer/counter register.

Discussion

The timer/counter register 0 (TM0) and the timer/counter register 1 (TM1) are used in operations requiring timing operations or event counters. The TM*n* command displays or changes the contents of the register. When *expr* is used, it is evaluated to a 16-bit value.

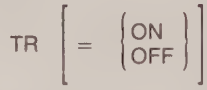
Examples

Key-in Sequence	Comments
TM0 RETURN	Displays the current contents of timer/counter register 0. The numeric value is displayed according to the default base.
TM0 = 100H RETURN	Changes the contents of timer/counter register 0 to the value 100H.
TM1 = 100H RETURN	Changes the contents of timer/counter register 1 to the value 100H.

TR

Enables or disables the display of general purpose registers

Syntax



Discussion

The register display (TR) command enables or disables the display of general purpose registers R0 through R7, the accumulator (ACC) at a breakpoint or during STEP emulation mode, and the PSW. If enabled, the registers are displayed on one line of screen.

Examples

Key-in Sequence	Comments
TR RETURN	Displays the current condition of register display.
TR=ON RETURN	Enables the display of general purpose registers R0 through R7, the accumulator (ACC), and the PSW.

TR*n*

Enables or disables the trace display

Syntax

TR*n* [= *expr* { ON / OFF }]

Where:

n	specifies up to four (0, 1, 2, or 3) program locations.
expr	specifies a memory address.

Discussion

The trace display command (TR*n*) enables or disables the trace display that has been previously specified with the TD, TR*n*, TB*n*, and TS commands, depending on program control passing to a specified memory address. Trace will begin at the specified memory address if that address is the first byte of an instruction. Up to four program locations can be specified.

Examples

Key-in Sequence	Comments
TR0 = 100H ON RETURN	Enables the trace display if the instruction at address 100H also has a breakpoint set.
TR0 RETURN	Displays the current condition of TR0.
TR1 = 110H OFF RETURN	Disables the trace display at address 110H. The trace information is enabled if the instruction at address 100H is also the location of a breakpoint. In STEP mode, the trace information is displayed for each instruction executed until program control reaches address 110H. In GO emulation, the only instructions displayed are those that also have breakpoints set. At address 110H, the trace display is disabled, and no further trace information is displayed until the display is re-enabled.

TS

Enables or disables the display
of the program status word register

Syntax

TS [= { ON / OFF }]

Discussion

The trace status display (TS) command enables or disables the display of the program status word (PSW) register when a breakpoint occurs. The register contains seven 1-bit flags that indicate the current state of the following items:

- Carry flag
- Auxiliary carry flag
- User flag 0
- Register bank select RS0 and RS1
- Overflow flag
- Parity flag
- UTL flag (reserved utility flag not accessible to the user)

Examples

Key-in Sequence	Comments
TS = ON RETURN	Enables the display of the PSW register on one line of the screen display.
TS = OFF RETURN	Disables the display of the PSW.

TV

Enables or disables the trace display

Syntax

$$\text{TV}\ \left[\ =\ \left\{\begin{matrix}\text{R}n\\ \text{ACC}\end{matrix}\right\}\ \textit{expr}\ \left\{\begin{matrix}\text{ON}\\ \text{OFF}\end{matrix}\right\}\ \right]$$

Where:

R*n*	specifies registers R0 throught R7.
expr	specifies a value that will be contained in either the specified register or the accumulator.

Discussion

The TV command enables or disables the trace display that is previously specified by the TD, TRn, TBn, and the TS commands, depending on the contents of a specified register. The user can specify any of the registers R0 through R7 or the accumulator (ACC).

NOTE

If the software trace display is turned off with either the TR*n* or TV commands, the display will remain off permanently until one of the commands which is set to turn on the display is executed, or until the EMV-51A software is reset.

Examples

Key-in Sequence	Comments
TV = R0 44H ON RETURN	Enables the trace display if general purpose register R0 contains 44H. The trace information is displayed if general purpose register R0 contains the value 44H, and will continue to be displayed for each instruction executed until a condition causes trace to be disabled.
TV = ACC 10H OFF RETURN	Disables the trace display if the accumulator contains 10H. No further trace informaton will be displayed until the user enables the display.

WRITE

Evaluates an expression and displays the results on the screen

Syntax

WRITE { *string* / *expr* } [,*string* / ,*expr*] ...

Where:

string	is a sequence of alphanumeric characters.
expr	specifies an arithmetic expression or numeric value. If an expression is used, it is evaluated and replaced by a 16-bit result.

Discussion

The WRITE command is used primarily inside of macro definitions to display messages on the screen. The ,*string* consists of a message that is displayed when the WRITE command is executed.

Example

Key-in Sequence	Comments
WRITE 'BOARD''S BAD' RETURN	Displays the message BOARD'S BAD. The double apostrophe is displayed as an apostrophe.

(macro execution)

Executes macro definitions

Syntax

:*string* [*parameter1, parameter2,...*]

Where:

:string	is a pre-defined macro name, where the name consists of 2 to 32 alphanumeric characters including the colon.
parameter	specifies any parameter values required within the macro definition.

Discussion

The user executes macro definitions by preceding the macro name with the colon (:).

If macro expansion is enabled with the ENABLE EXPANSION command, the macro definition is listed prior to execution of the macro.

If the macro definition requires parameters, the parameters are entered with the macro name. The parameters are entered as follows:

parameter 1,parameter 2,...

The parameters sequentially replace the parameter place holder within the definition. Refer to Chapter 6 for additional details on using parameters within macros.

Examples

Key-in Sequence	Comments
:SUM RETURN	Executes the macro :SUM
:DEMO 123,12,.ABLE RETURN	Executes the macro :DEMO after substituting the parameters 123, 12, and .ABLE into the macro definition.

(symbol handling)

Displays both system and user symbol names or modifies user symbol names

Syntax

.string[= *expr*]

Where:

.string	is a pre-defined symbolic name consisting of 2 to 32 alphanumeric characters including the period.
expr	specifies the value to be assigned to the symbolic name. It consists of numeric values, symbol names, and arithmetic operators and is evaluated to a 16-bit result.

Discussion

The emulator software or the user can assign a symbolic name to a specified value. Both the system symbolic names and the user symbolic names are preceded by a period. The emulator software responds to symbolic names by substituting the value assigned to the name for the name itself.

If the symbolic name is a user symbolic name, the = *expr* command modifier is used to change the value assigned to the name.

Examples

Key-in Sequence	Comments
.ABLE RETURN	Displays the value assigned to the symbolic name .ABLE.
.CY RETURN	Displays the current contents of the system carry flag.
.ABLE = 123H RETURN	Changes the contents of user symbolic name .ABLE to the value 123H.

APPENDIX A
ERROR MESSAGES

This appendix contains a list of the error and warning messages produced by the emulator software.

ERR 22:COPROCESSOR HAS FILE
An attempt was made to open or write to a file currently being written to by the other processor.

ERR 24:TOO MANY BREAKS
Too many breakpoints are set for this command to operate. Delete some of the breakpoints and re-enter the command.

ERR 25:RELOCATABLE FILE
The command attempted to load a relocatable object file. Use the assembler or the linker to convert the file to absolute format, then repeat the command.

ERR 26:UNRESOLVED EXTERNALS
The command atted to load a file with unresolved symbols. Use the assembler or the linker to resolve the external symbol or name, then repeat the command.

ERR 52:UNWRITEABLE MEMORY
The last command attempted to write to external memory. Check the command to ensure writing has not been attempted above FFH.

ERR 80:SYNTAX ERROR
The command is ignored. Check the command syntax and re-enter the command.

ERR 81:INVALID TOKEN
A token in the previous line is not properly formed, or it is not appropriate for the command context; for example, the entry CBYTE 300 when SUFFIX = Y (binary).

ERR 83:INAPPROPRIATE NUMBER
Too many values were used for a specified range.

ERR 85:ITEM ALREADY EXISTS
The previous DEFINE command referred either to a symbol already in the user's or system symbol table, or to a macro already in the macro definition table.

ERR 86:ITEM DOES NOT EXIST
The symbol or macro reference in the previous command is not defined.

1. For a symbol, check the version of the code loaded, or define the symbol in the current session.
2. For a macro, define or INCLUDE the macro definition in the current session.

ERR 88:MACRO PARAMETER ERROR
A macro call contained more than 10 actual parameters, or a parameter contained too many characters. Check the definition of the macro.

ERR 89:MISSING CR-LF IN FILE
The current INCLUDE file does not end with a carriage return/line feed. Exit and use the editor to correct the file.

ERR 8F:NON-NULL STRING NEEDED

A null string (apostrophes with no enclosed characters) was used where at least one character is required, such as DBYTE 1 = ''.

ERR 90:MEMORY OVERFLOW

The emulator workspace exceeded the amount allocated to it. The workspace contains the user symbol table and space for expansion of macros prior to their execution. The command that produced the overflow is aborted, but the memory already written remains as written. To reclaim workspace, remove some user symbols.

ERR 92:COMMAND TOO LONG

The command exceeds the capacity of the emulator's command buffer; possibly caused by too many operators. Break the command or expression into several smaller units.

ERR 95:INVALID OBJECT FILE

The object file referenced in a LOAD command is not written in the proper format. The file may be a text file rather than an object file.

1. Select another file for loading.
2. EXIT and verify the file type.

ERR 99:EXCESSIVE ITERATED DATA

The number of data items to be repeated in memory exceeds the buffer size for iterated data (128 bytes) (See the following example.)

Example **CBYTE 30 TO 1000H = RBYTE 0 TO 256T**

ERR 9D:LINE TOO LONG

The input line exceeds 122 characters.

1. Use continuation lines (ampersand at the end of the line to identify intermediate carriage return) to divide the command line into several input lines.
2. Break command into two or more shorter command lines.

ERR A4:MACRO FILE FULL

The temporary file MAC.TMA has used all the available space on the disk specified at invocation as the WORKFILE disk. Save and remove macro definitions to make room for more, using the PUT and REMOVE commands.

ERR B3:OFFSET IS TOO LARGE

This error occurs when the expression specified within an assembly mnemonic instruction results in a relative offset that is larger than 8 bits.

ERR B9:NO HELP AVAILABLE

There is no help message for the item requested.

ERR BC:SYSTEM SYMBOL ERROR

Illegal operation on the system type symbols, such as attempting to change the value of a system symbol or using a multiple reference involving system symbols.

ERR E7:ILLEGAL FILENAME

The command specified a filename that does not conform to ISIS-iPDS format specifications. Check the *iPDS™ User's Guide* for the proper filename format.

ERR E8:ILLEGAL DEVICE

The command contained a reference to an ISIS-iPDS device, but the reference is illegal or unrecognizable. Check the *iPDS™ User's Guide* for the proper device names.

ERR F0:NO SUCH FILE

The command specified a file that does not exist on the designated disk.

1. Verify the drive number and filename.
2. Check to see that the correct disk is inserted in the drive.

ERR F1:WRITE-PROTECTED FILE

The command attempted to open a write-protected file for write access.

1. Select another file.
2. EXIT and remove write protection from the target file (refer to the *iPDS™ User's Guide* for details).

ERR F3:CHECKSUM ERROR

A checksum error in an object file was encountered during loading.

1. Select another file.
2. EXIT and create a correct object file.

ERR F6:DISKETTE FILE REQUIRED

The command specified a device other than a disk file, but the operation requires a disk file.

ERR F9:ILLEGAL ACCESS

The command attempted to open a read-only device for write access (e.g., :CI: as a LIST device), or attempted to open a write-only device for read access (e.g., :LP: in LOAD command). Check the command syntax for a list of appropriate devices.

ERR FA:NO FILE NAME

The command references a disk device, but omitted the filename; for example: LOAD :F1:

ERR FF:NULL FILE EXTENSION

The command referenced a file terminated by a period, but the implied extension is missing. Omit the period or include the extension (refer to the *iPDS™ User's Guide* for details on filenames and extensions).

WARNING D0

The instruction at the brakpoint was fetched but noan interrupt r befthe instruction. The breakpoint instruction is the first to ee afterinterrupt service routine.

WARNING D1

A breakpoint occurs on the first instruction of an interrupt service routine. The break instruction executed normally, but the previous instruction did not execute. The interrupted instruction is the first instruction to execute when emulation resumes. Check for a breakpoint set at the address of the first instruction of the interrupt service routine.

WARN D2

This warning is displayed when beginning GO or STEP emulation and a timer is enabled with an initial value of 0 or 1. This condition generates an erroneous timer overflow and forces an interrupt if interrupts are enabled.

WRN CB:TRUNCATED TO 8 BITS

This error occurs when an opcode, value, or an expression in a mnemonic instruction to be assembled is greater than 8 bits. The value is truncated to the low 8 bits, the warning is issued, and processing continues.

WARN CD:TRUNCATED TO 11 BITS

This error occurs when an expression value in the ACALL or AJMP instruction in assembly is larger than 11 bits. The value is truncated to 11 bits, the warning error message is issued, and processing continues.

APPENDIX B
CONFIDENCE TESTS

Confidence Test

This section describes the operation of the confidence test for the EMV-51A emulation vehicle. It is not necessary to run this test prior to using the system. However, it is recommended that the test be run when the emulator is initially installed.

The confidence test aids in troubleshooting the system if problems occur. Successful execution of the test demonstrates the complete operation of the system.

The confidence test assumes that the ISIS-iPDS operating system is loaded and running. The test further assumes that the EMV-51A emulator hardware is installed. The test runs as a utility under the ISIS-iPDS operating system and provides a set of subcommands that help verify the emulator.

To perform the confidence test:

1. Load the confidence test command under the ISIS-iPDS operating system by entering the PCONF command line.

2. Initialize the confidence test with the INIT E51CON command.

3. Enter any of the nine confidence test commands to perform the 15 different tests.

These steps are described in detail in the following sections. Notational conventions used throughout for command descriptions are introduced and explained in Chapter 10.

PCONF Command

The PCONF command is shown in the following example:

```
A0>PCONF
ISIS-PDS PCONF Vx.y
*
```

The PCONF command is entered when the ISIS-iPDS prompt is displayed indicating that the operating system is ready to accept a command. It loads the confidence test program which displays a sign-on message and an asterisk prompt allows the entry of the INIT E51CON command.

INIT E51CON Command

The INIT E51CON command produces the following display:

```
*INIT E51CON
EMV-51 CONFIDENCE TESTS, Vx.y
USER RETURN
```

After the INIT E51CON command is entered, another sign-on message and asterisk prompt are displayed. Enter any of the nine confidence test commands.

Confidence Test Commands

The nine confidence test commands allow you to specify the test sequence for the 15 confidence tests and to control the reporting of results from the tests. The nine commands are as follows:

Command	Abbreviation
CLEAR	CLE
DESCRIBE	DES
ERROR	ERR
EXIT	EXI
IGNORE	IGN
LIST	LIS
RECOGNIZE	REC
SUMMARY	SUM
TEST	TES or T

Any command consisting of four or more letters can be abbreviated to the first three letters. Additionally, the TEST command can be abbreviated to T. The TEST command is the command used to run the confidence tests.

Each of these commands is described in one of the following sections. The 15 confidence tests are described under the TEST command.

CLEAR

Syntax

CLEAR { *test number* [,*test number*] . . . | *test number* TO *test number* }

Where:

test number specifies one of the 15 confidence tests by number (00H through 0EH). Enter the hexadecimal value of the test number.

Discussion

The CLEAR command sets the execution count and error count to zero for the test or tests specified. The execution count is the number of times the specified test or tests have been run and the error count is the number of errors detected. If no tests are specified, the CLEAR command clears the counts for all the tests. The CLEAR command does not affect the status of a test. The status of a test indicates whether the test is ignored or recognized. See the IGNORE and RECOGNIZE commands.

Example

Key-in Sequence		Comments
CLEAR 3,4,5	RETURN	Clears the execution count and error count for tests 3, 4, and 5.

DESCRIBE

Syntax

DESCRIBE { *test number* [,*test number*] . . . | *test number* TO *test number* }

Where:

test number specifies one of the 15 confidence tests by number (00H through 0EH). Enter the hexadecimal value of the test number.

Discussion

The DESCRIBE command displays the name and the status of the test or tests specified. The status of a test indicates whether the test is ignored or recognized by the system. See the IGNORE and RECOGNIZE commands. If no tests are specified, the name and status are given for all the tests.

Example

Key-in Sequence	Comments
DESCRIBE 3 TO 6 RETURN	Describes tests 3, 4, 5, and 6, when test 3 is being ignored by the system, as shown in the following screen display:

```
00003H  BREAK REGISTER MEMORY TEST**** IGNORED ****
00004H  BREAK/COMMAND MULTIPLEXER TEST
00005H  RUNNING/INTD STATUS TEST
00006H  BREAK SYNCHRONIZATION TEST
```

Note that test numbers are displayed in hexadecimal.

ERROR

Syntax

ERROR = [*n*]

Where:

n is either 0 to display pass/fail messages or 1 to suppress them. The default setting is 0 to display all pass/fail messages.

Examples

Key-in Sequence	Comments
ERROR RETURN	Displays the current status.
ERROR = 1 RETURN	Suppresses pass/fail messages for the confidence tests.

EXIT

Syntax

EXIT

The EXIT command ends the test session and returns control to the ISIS-iPDS operating system.

IGNORE

Syntax

IGNORE [*test number* [,*test number*] . . .
test number TO *test number*]

Where:

test number specifies one of the 15 confidence tests by number (00H through 0EH). Enter the hexadecimal value of the test number.

Discussion

The IGNORE command allows declaring the specified test or tests to be ignored and not run.

Example

Key-in Sequence	Comments
IGNORE 5,6,8 RETURN	Runs all tests except 5, 6, and 8.

LIST

Syntax

LIST *destination*

Where:

destination specifies a valid ISIS-iPDS device to receive any output from the tests. See Chapter 10 for an explanation of destination.

Discussion

The LIST command causes a copy of all subsequent output from the tests to be sent to *destination*. The output includes prompts, user entered input, and error messages. The output is also displayed on the display screen. If the *destination* is :CO:, there is no effect, since the console receives all the output anyway.

Example

Key-in Sequence	Comments
LIST :LP: RETURN	Copies all output to the line printer.

RECOGNIZE

Syntax

RECOGNIZE [*test number* [,*test number*] . . .
test number TO *test number*]

Where:

test number specifies one of the 15 confidence tests by number (00H through 0EH). Enter the hexadecimal value of the test number.

Discussion

The RECOGNIZE command allows declaring of the specified test or tests to be recognized and run.

Example

Key-in Sequence	Comments
RECOGNIZE 5,7,8 RETURN	Assuming that tests 5, 6, 7, and 8 are currently ignored, this command will recognize 5, 7, and 8. Test 6 continues to be ignored.

SUMMARY

Syntax

SUMMARY [*test number* [,*test number*] . . .] [E0]
[*test number* TO *test number*]

Where:

test number	specifies one of the 15 confidence tests by number (00H through 0EH). Enter the hexadecimal value of the test number.
E0	specifies that only the tests with a non-zero error count will be summarized.

Discussion

The SUMMARY command displays the following information for each test or tests specified:

- Test number
- Number of times the test was executed
- Number of times an error occurred during the test
- Status of the test (ignored or recognized)

The information displayed is accumulated since the last INIT E51CON or CLEAR command. If no test or tests are specified, summaries are displayed for all tests. If EO is specified, a summary is displayed only for those tests with a non-zero error count.

Examples

Key-in Sequence	Comments
SUMMARY 3 TO 5	Displays a summary of tests 3, 4, and 5. The following display will appear on the screen (note that all test numbers and error counts are given in hexadecimal).

```
00003H LINE PRINTER   TEST EXECUTE 00002H TIMES, 00000H
                      FAILURES
00004H BREAK/COMMAND  TEST EXECUTE 00002H TIMES, 00001H
                      FAILURES
00005H RUNNING/INTD   TEST EXECUTE 00002H TIMES, 00000H
                      FAILURES
```

Key-in Sequence	Comments
SUMMARY 3 TO 5 EO RETURN	Displays a summary of tests 3, 4, and 5 only if an error is detected. The following display will appear on the screen:

```
00004H BREAK/COMMAND  TEST 00002H TIMES, 00001H FAILURES
```

TEST

Syntax

TEST { [test number [,*test number*] . . . | *test number* TO *test number*] } { ON ERROR | ON NOERROR | COUNT *nnnn* | FOREVER }

Where:

test number	specifies one of the 15 confidence tests by number (00H through 0EH). Enter the hexadecimal value of the test number.
nnnn	specifies the number of times the specified test or tests are to be run. The value can be given in hexadecimal (H), decimal (T), octal (Q), or binary (Y) by attaching the appropriate suffix to the value. The default base is hexadecimal. The maximum value of *nnnn* is 65,535 in decimal.
ON ERROR	specifies that the tests execute repeatedly if one or more errors are detected.
ON NOERROR	specifies that the tests execute repeatedly if no errors are detected.
COUNT *nnnn*	specifies that the tests execute in numerical order for *nnnn* times. If *nnnn*=0 or if COUNT is specified with no *nnnn*, no tests execute.
FOREVER	specifies that the tests execute in numerical order and repeat until the the ESC key is pressed.

Discussion

The TEST command loads and runs the test or tests specified. The tests are executed in numerical order regardless of the order in which they were entered. If no test number is specified, all currently recognized tests are executed.

Examples

Key-in Sequence	Comments
TEST RETURN	Runs all tests.
TEST 0 TO 0E RETURN	Runs tests 0 through 0EH. Note that the value 0E is a hexadecimal value.
TEST 4,9,2,8 RETURN	Runs tests 2, 4, 8, and 9. Note that the commands are loaded and run in numerical order.
TEST 0 TO 8 ON ERROR RETURN	Runs tests 0 through 8 and repeats the failing test if an error is detected.
TEST 7 ON NOERROR RETURN	Runs test 7 and repeats if no error is detected.
TEST 3,5 FOREVER RETURN	Runs tests 3 and 5 and repeats forever.
TEST 2,7 COUNT 5 RETURN	Runs tests 2 and 7 and repeats 5 times.

Confidence Tests

The following tests are available with the EMV51A diagnostic software:

Test Number	Function
0	Program memory test
1	Command memory test
2	Break register memory test
3	Break/command multiplexer test
4	RUNNING/INTD status test
5	Break synchronization test
6	Break detection inputs
7	Address line test
8	Command sequence end detection
9	Interrogation INTR1 test
A	Program memory data path
B	MOVC logic test
C	Interrupt at break
D	Instruction set test
E	Miscellaneous processor tests

Almost all of these tests require some interaction while running. If a response is not received in a predetermined period of time, the test times out and is not executed.

Each of these tests is described in the following sections.

Test 0 - Program Memory Test
Test 0 verifies the address uniqueness and data integrity of the program storage memory.

Test 1 - Command Memory Test
Test 1 verifies the address uniqueness and data integrity of the command sequence memory.

Test 2 - Break Register Memory Test
Test 2 verifies the address uniqueness and data integrity of the break register memory.

Test 3 - Break/Command Multiplexer Test
Test 3 verifies the operation of the address line multiplexer into the break and command memory.

Test 4 - RUNNING/INTD Status Test
Test 4 verifies that the RUNNING and the INTD status bits can be set and cleared with control procedures. It verifies that the proper idle state data is sent to the break address register.

Test 5 - Break Synchronization Test
Test 5 verifies that the breakpoint detection circuit and the stop emulation circuit stop the emulation. This test fills the break register with data to break at any location executed.

Test 6 - Break Detection Inputs
Test 6 verifies that each of the four pairs of beak register data in the break detector sets the address compare line properly.

Test 7 - Address Line Test
This test verifies the address line data in the break address register.

Test 8 - Command Sequence End Detector
Test 8 verifies that asserting address line 8 under command sequence control stops the sequence and returns the unit to the IDLE mode.

Test 9 - Interrogation INTR1 Test
This test verifies that er wrgeral purpose emulator register in a command sequence the correct data is brought out of the processor when the INTER line is asserted in the control register.

Test A - Program Memory Data Path
This test single steps a two-byte instruction (AA,55 - MOV R2,55) to verify the program data path to the processor.

Test B - MOVC Logic Test
Test B verifies the command sequence to fetch a byte of code data from program memory location with the address line number 11 asserted and not asserted.

Test C - Interrupt at Break
Test C verifies the operation of the simultaneous interrupt along with a breakpoint logic by emulating a routine that forces a timer interrupt to occur with an emulation break.

Test D - Instruction Set Test
This test loads an 8051 instruction set test into program memory and executes it.

Test E - Miscellaneous Processor Tests
Test E loads a routine that verifies the operation of the microprocessors internal RAM, timers, interrupts, and serial port.

Confidence Test Error Messages

CONTROLLER FAILED TO INITIALIZE
STATUS EXPECTED: 05H, ACTUAL: xxH
After writing to the control register 20 times, the controller failed to reset to the initial idle state.

PROGRAM
COMMANDMEMORY FAILED
BREAK REGISTER
DATA EXPECTED: eeH, ACTUAL: aaH, LOCATION: xxxxH
While testing one of the memory sets, a data or addressing error occurred at location (xxxxH). The expected data (eeH) did not match the actual data (aaH).

BREAK/COMMAND MEMORY MULTIPLEXER FAILED
SELECTOR STUCK LOW
SELECTOR STUCK HIGH
LOW NIBBLE FAILED TO SELECT
HIGH NIBBLE FAILED TO SELECT
Either the multiplexer select line did not change states or one of the multiplexers is failing to change states.

RUNNING FAILED TO SET
INTD CLEAR
Control output sequences failed to cause one of the status inputs to change properly.

IDLE BREAK ADDRESS FAILED
EXP: 0202, ACT: xxxxH
The break address did not contain the proper address after putting the processor into an idle state.

BREAK SYNCHRONIZATION FAILED
BREAK ADDR-EXP:0000H, ACT:
BREAK ADDR-EXP: 0000H, ACT: xxxxH
NEXT ADDR-EXP : 0001H, ACT: yyyyH
The break address register did not contain the right data after setting the break register to break on any instruction and to execute a NOP instruction at location 0.

BREAK DETECTOR FAILED WITH 11H AS INPUTS
22H
44H
88H
One of the four pairs of inputs into the break detector failed to trigger a break sequence.

ADDRESS LINE FAILURE
EXP:AAAAH, ACT:
EXP: AAAAH, ACT: xxxxH
5555H
The break address did not contain the proper data after executing a single step sequence to location AAAAH or 5555H.

END OF COMMAND SEQUENCE FAILED
In a command sequence, a jump to a trigger address failed to set the CMDEN bit in the status register.

INTR1 DATA FAILED
EXPECTED: AAH, ACT: xxH
The data failed to match after loading the general purpose EMV register in the processor with a command sequence and trying to read it back by asserting the INTER signal in the control register.

PROGRAM MEMORY DATA PATH FAILED
R2 DATA - EXP: 55H, ACT: xxH
Register R2 failed to contain a value of 55H after loading location 55H with a 55H and R2 with 00 and executing a sequence (AAH, 55H - MOV R2, 55H).

MOVC SEQUENCE FAILED AT 0000H
0800H
The execution of a MOVC failed to fetch the proper data.

INTERRUPT DURING BREAK FAILED
The status bit failed to indicate that executing a program sequence had caused an interrupt during break.

INSTRUCTION SET FAILURE IN GROUP # xx
The processor failed the instruction set test.

TIMER x FAILED
INTERNAL RAM FAILURE
SERIAL PORT OUTPUT FAILED
These are errors that occurred while testing the internal functions of the 8051 processor.

INDEX

ACCUMULATOR command, 10-7
  Tutorial, 6-2
Address breakpoints, 7-1, 7-3, 10-15
  Tutorial, 2-6, 7-3, 7-11
Advanced Commands, 4-1, 8-1 thru 8-15
Architecture overview, 1-1
Arithmetic operators, 9-2
ASM command, 6-6 thru 6-7, 10-8

Base, 5-10, 10-5, 10-10, 10-63
BASE command, 5-10, 10-10
BC command, 7-2, 7-3, 10-11
Branch breakpoints, 7-3, 10-12
BRB command, 7-2, 10-12
  Tutorial, 7-10
BREAK command, 2-8, 7-3, 10-13
  Tutorial, 2-8, 7-3, 7-11 thru 7-12
Breakpoint restrictions, 7-2
B command, 10-9
B register, 10-9
BRR command, 7-2, 10-14
Br <n> command, 7-1, 10-15
BV command, 7-2, 10-16

CBYTE command, 6-4, 6-5, 9-2 thru 9-3, 10-17
CBYTE restrictions, 9-3
CDUMP command, 6-5, 10-18
CLEAR command, B-2
Clearing breakpoints, 7-2, 10-11
Clock, 3-1, 3-2
Code memory, 9-2 thru 9-3
Command categories, 4-1
Command files, 8-4 thru 8-5
Command line, 4-2 thru 4-3, 10-1, 10-2
Command line editing, 4-2 thru 4-3, 10-2, 10-3
Command line execution, 4-2, 10-2
Comment lines, 4-3
Compound commands, 8-8 thru 8-12
Conditional IF commands, 8-8 thru 8-9, 10-36 thru 10-38
Confidence test, B-1
Confidence test commands, B-2 thru B-6
Confidence test error messages, 13-9
Content operators, 9-2 thru 9-3
Continuation lines, 4-2, 10-2
Control characters, 4-2 thru 4-3, 10-2 thru 10-3
Controller, 1-1, 1-2, 3-1 thru 3-2
Controlling the display, 4-3, 10-10
COUNT command, 8-8, 8-10 thru 8-11, 10-19
CTRL, 4-3, 10-2
CTRL Q, 4-3, 10-3
  Tutorial, 2-9
CTRL R, 4-2, 10-2
CTRL S, 4-3, 10-3
  Tutorial, 2-9
CTRL X, 4-3, 10-2
CTRL Z, 4-3, 10-3

DASM command, 2-8 thru 2-9, 4-7, 10-21
Data pointer register (DPTR), 6-1, 10-27
DBYTE command, 6-4, 8-2, 10-22
DBYTE content operator, 9-3
DDUMP command, 6-5, 10-23
DEFINE command, 5-7, 8-1, 8-2, 10-24
DESCRIBE command, B-2 thru B-3
Designing the product, 1-5
Designing with the 8051, 1-3
Destination, 10-6
DIR command, 8-3, 10-25
DISABLE command, 5-9, 8-6, 10-26
DISABLE EXPANSION, 8-6, 10-26
Display/modify commands, 4-1, 6-1 thru 6-8
Displaying register contents, 6-1
DTRACE command, 7-7 thru 7-8, 10-28

EA/jumper, 6-5, 3-3
Editing, 4-2, 10-2 thru 10-3
EM command, 8-2 thru 8-3
Emulation
  Definition, 1-5
  Procedure 1-6 thru 1-7
  With emulation, 1-6
  Without emulation, 1-6
Emulation commands, 4-1
Emulation session, 1-6 thru 1-7, 2-4 thru 2-8
Emulator 1-4, 3-2
EMUL signal, 3-1
EMV-51A components
  Hardware, 1-2 thru 1-4, 3-1 thru 3-2
  Publications, 1-5
  Software, 1-4
EMV-51A demonstration program
  Listing, 2-7
EMV-51A invocation, 2-5
ENABLE command, 5-8 thru 5-9, 10-29
Enable expansion, 86
ERROR command, B-3
ESC, 4-2, 10-2
EVALUATE command, 5-11, 10-30
EXIT command, 2-10, 5-2, 10-31, B-4
EXTBRK signal, 3-1

EXT10, 6-4
EXT11, 6-4
Expression, 10-4 thru 10-5
External data meory, 9-3

FUNCTION command, 8-11 thru 8-12, 10-32 thru 10-33
FUNCT S, 4-3
FUNCT 2, 7-4
FUNCT 3, 7-8
Function keys, 8-11 thru 8-12

GO command, 7-9, 10-34
  Tutorial, 2-10

Hardware, 1-2, 3-1 thru 3-3
HELP command, 10-35
  Tutorial, 5-2 thru 5-3
Help information, 5-2

IF..THEN..ELSE, 8-8 thru 8-9, 10-36 thru 10-38
IGNORE command, B-5
INCLUDE command, 8-4, 8-7, 10-39
INI51CON command, B-1
Installation procedures
  Choosing 4K or 8K RAM, 2-1
  Controller, 2-1, 2-3
  Emulator module, 2-4
  EMV-51A software, 2-4
INTERRUPT command, 6-3, 10-40
Interrupt enable, 6-3
Interrupt priority, 6-3
Interrupts in progress, 6-3
Item, 10-4

Jumper configurations, 3-3
LIST command, 5-5, 10-41, B-5
List file, 5-5
LOAD command, 2-6, 5-4, 10-42
Loading programs, 2-6, 5-4
Loading multiple programs, 5-4
Logical operators, 9-2, 9-6

Macros, 4-3, 8-1 thru 8-19
MACRO command, 8-4, 10-43, 10-73
  Tutorial, 8-4
Macro definitins, 8-1 thru 8-3
Macro execution, 8-4, 10-73
Manipulating symbols, 5-6, 5-7 thru 5-8
MEMORY command, 6-4 thru 6-6, 10-44
Memory contents, 6-4 thru 6-5
Messages, 4-3
Modify commands, 4-1
Multiplication register, 9-5

Notational conventions, 10-3 thru 10-4

ON ERROR, B-8
ON NOERROR, B-8
Operators, 9-2, 9-6

Parameters, 8-2
P command, 7-10, 10-45
PBYTE command, 10-46
PC, 6-1, 10-47
PCONF command, B-1
Port 0, 3-2
Prompts, 4-3
PSENE signal, 3-1
PSW, 10-48
PUT command, 8-4 thru 8-5, 10-49

Range breakpoints, 7-2
RBIT command, 10-50
RBS command, 10-51
RBYTE command, 10-52
RBYTE content operator, 9-3
RECOGNIZE command, B-6
REGISTER command, 6-2, 10-53
Register contents
  Displaying, 6-2
  Displaying INTERRUPT, 6-3
  Modifying, 6-1
Register keyword names, 6-1
Register value breakpoints, 7-2
Relational operators, 9-2, 9-6
REMOVE command, 5-8, 8-6 thru 8-7, 10-55
REPEAT command, 8-9 thru 8-10, 10-56 thru 10-59
RESET command, 5-10, 10-58
Reset input, 3-3
RETURN, 4-2, 10-1
Rn, 6-2, 10-54
RUBOUT, 4-2, 10-2

SAVE command, 5-4, 10-59
  Tutorial, 2-10
Saving programs, 2-10, 5-4
Setting breakpoints, 7-1
SINT, 6-4
Software, 1-4, 2-4 thru 2-6
Source, 10-6
SP command, 6-1, 10-60
STEP command, 7-8 thru 7-9, 10-61 thru 10-62
String, 10-4
Suffix, 5-10
SUFFIX command, 5-10, 10-5, 10-63
SUMMARY command, B-7 thru B-8
Supply voltage, 3-4
SYMBOLS command, 5-8, 10-64

Symbol handling, 10-74
Symbolic reference
  Assigning symbol names, 5-6
  Displaying symbol name, 5-7
  Manipulating symbols, 5-6
  Removing symbol names, 5-8

TBn command, 7-5, 10-65
TD command, 7-5, 10-66
Test 0, B-9
Test 1, B-10
Test 2, B-10
Test 3, B-10
Test 4, B-10
Test 5, B-10
Test 6, B-10
Test 7, B-10
Test 8, B-10
Test 9, B-10
Test A, B-10
Test B, B-10
Test C, B-10
Test D, B-11
Test E, B-11
TMn, 10-67
Timer0, 6-4
Timer1, 6-4
TR command, 7-5, 10-68
Trace, 7-1, 7-4 thru 7-7, 10-68
TRn command, 7-6, 10-69
TS command, 7-6, 10-70
TV command, 7-6 thru 7-7, 10-71

UPPERCASE, 10-3
User publications, 1-5
User test signals, 1-2 thru 1-3
Utility commands, 4-1, 5-1 thru 5-11

Workfiles, 10-1 thru 10-2
WRITE command, 8-2 thru 8-3, 10-1, 10-72

REQUEST FOR READER'S COMMENTS

Intel's Technical Publications Departments attempt to provide publications that meet the needs of all Intel product users. This form lets you participate directly in the publication process. Your comments will help us correct and improve our publications. Please take a few minutes to respond.

Please restrict your comments to the usability, accuracy, readability, organization, and completeness of this publication. If you have any comments on the product that this publication describes, please contact your Intel representative. If you wish to order publications, contact the Intel Literature Department (see page ii of this manual).

1. Please describe any errors you found in this publication (include page number).

2. Does the publication cover the information you expected or required? Please make suggestions for improvement.

3. Is this the right type of publication for your needs? Is it at the right level? What other types of publications are needed?

4. Did you have any difficulty understanding descriptions or wording? Where?

5. Please rate this publication on a scale of 1 to 5 (5 being the best rating). _____

NAME _____ DATE _____

TITLE _____

COMPANY NAME/DEPARTMENT _____

ADDRESS _____

CITY _____ STATE _____ ZIP CODE _____
(COUNTRY)

Please check here if you require a written reply. ☐

WE'D LIKE YOUR COMMENTS ...

This document is one of a series describing Intel products. Your comments on the back of this form will help us produce better manuals. Each reply will be carefully reviewed by the responsible person. All comments and suggestions become the property of Intel Corporation.

NO POSTAGE
NECESSARY
IF MAILED
IN U.S.A.

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 79 BEAVERTON, OR

POSTAGE WILL BE PAID BY ADDRESSEE

Intel Corporation
5200 N.E. Elam Young Parkway.
Hillsboro, Oregon 97123

DSHO Technical Publications

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, California 95051 (408) 987-8080

Printed in U.S.A.

DEVELOPMENT SYSTEMS

www.manualsplus.com

1033391

0120/4K/1284/AP/AD